L STRATER (1818) BAR	50 183	TION PA	GE		OMB No. 0704-0188
		erage Coolar ber roy the Control of String Massington Heada Management and Sy	consel including the til rimation – Send Comme Jarters Services, Direct oget – Paperwork Rieduc	mo for reviewing inst ints regarding this bu orate for informatio rulin Project (0704-01	ructions, search no existing dota sources, irden estimate or any other assect of this n Operations also Reports in 2.15 Jefferson 88) Washington, LiG 20502
AGENCY USE ONLY (Leave DI	iank) 2. REPORT D	ATE	3. REPORT TYP	E AND DATES	COVERED
TITLE AND SUBTITLE			1 11101	5. FUNI	DING NUMBERS
Execution lime Pre	diction of Ada	i Programs			
AUTHOR(S)					
Christopher A. War	ack, Captain				
PERFORMING ORGANIZATION	NAME(S) AND ADDR	ESS(ES)		B. PERF REPC	ORMING ORGANIZATION
AFIT Student Attendi	ing: Universit	y of Michi	gan	AFIT/	ci/cia-91-134
SPONSORING / MONITORING &	GENCY NAME(S) AND	ADDRESS(ES)			
AFTT/CT				AGE	NCY REPORT NUMBER
Wright-Patterson AFI	3 OH 45433-658	3	MAY	7 1992	
I. SUPPLEMENTARY NOTES	<u></u>				
			,		
Approved for Public Distributed Unlimite ERNEST A. HAYGOOD, Executive Officer	Release IAW l ed Captain, USAF	90–1			
5. ABSTRACT (Maximum 200 wc	ords)			L	**************************************
A. SUBJECT TERMS					15. NUMBER OF PAGES
. SUBJECT TERMS					15. NUMBER OF PAGES 121 16. F.V.CE CODE
SUBJECT TERMS	18. SECURITY CLAS OF THIS PAGE	SSIFICATION	9. SECURITY CL OF ABSTRAC	A SSIFICATION T	15. NUMBER OF PAGES 121 16. F.J.CE CODE 20. LIMITATION OF ABSTRAC

)	3	4
<u> </u>		1

011011
INSPECIEL
4

Acea	solon For	
NTIS	GRANI	N
174 0	TAB	ā
Unan	oursed	ā
Just	if isation_	
By		_
Dist	ibution/	
Avai	lability	Codes
	Aveil and	/or
Dist	Special	•
01	1 1	
14-1	1 1	
-	1	

Execution Time Prediction of Ada Programs

Christopher A. Warack Captain, US Air Force

1991

121 pages

Master of Science University of Michigan

Specification of the timing properties of real-time systems is a fundamental part of their requirements. Analyzing the timing properties of the system's design and implementation is an important issue for the system developer. Timing analysis is necessary to determine the validity of a design or implementation in respect to the real-time specification.

Using timing schema and PERT networks, Ada program timing behavior can be analyzed. The use of PERT networks is simple but restricted to single processor systems. Replacing the PERT networks with a communicating real-time state machines model allows the analysis of Ada programs on multi-processor systems.

The technique is developed with examples and applied to a Macintosh IIsi programming environment. A foundation is laid for measuring how good a timing analysis prediction fits the implementation.



92 5 01 013

Bibliography

- 1. Shaw, A.C., *Reasoning About Time in Higher-Level Language Software*. IEEE Transactions on Software Engineering, 1989. 15(7): p. 875 889.
- 2. Park, C.Y. and A.C. Shaw. A Source-Level Tool for Predicting Deterministic Execution Times of Programs . Department of Computer Science, University of Washington, (Technical Report 89-09-12). September 13, 1989.
- 3. Shaw, A.C. Towards a Timing Semantics For Programming Languages. in Third Annual Workshop, Foundations of Real-Time Computing. 1990. Washington, DC: Office of Naval Research.
- 4. Goos, G., W.A. Wulf, A. Evans Jr., and K.J. Butler, ed. *DIANA: An Intermediate Language for Ada*. Lecture Notes in Computer Science, ed. G. Goos and J. Hartmanis. 1983, Springer-Verlag: Berlin. 201 pages.
- 5. Shaw, A.C. Communiating Real-Time State Machines . Department of Computer Science and Engineering, University of Washington, (Technical Report 91-08-09). August 1991.
- 6. Hoare, C.A.R., An Axiomatic Basis for Computer Programming. Communications of the ACM, 1969. 12 (10): p. 576-580.
- 7. Shaw, M. A formal System for Specifying and Verifying Program Performance Department of Computer Science, Carnegie-Mellon University, (Technical Report CMU-CS-79-129). 21 June 1979.
- 8. Walden, E. and C.V. Ravishankar, A Survey of Hard Real-Time Scheduling Algorithms. unpublished draft, 1990.
- 9. Cornhill, D., et al. Limitations of Ada for Real-Time Scheduling. in Proceedings of the International Workshop of Real-Time Ada Issues. 1987. Moretonhampstead, Devon, UK: ACM SIGAda.
- 10. Military Standard Ada Programming Language , ANSI/MIL-STD-1815A, U.S. Department of Defense, Ada Joint Program Office, (January 1983).
- 11. Motorola, MC68030 Enhanced 32-Bit Microprocessor User's Manual . Third ed. 1990, Englewood Cliffs, New Jersey: Prentice-Hall, Inc.
- 12. Park, C.Y. and A.C. Shaw. Experiments With a Program Timing Tool Based on Source-Level Timing Schema. in IEEE Real-Time Systems Symposium . 1990. Lake Buena Vista, Florida: IEEE Computer Society Press.

Execution Time Prediction of Ada Programs

by Christopher Allen Warack

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science (Computer Science and Engineering) in The University of Michigan 1991

Thesis Committee:

Professor Kang G. Shin, Chairman Professor C.V. Ravishankar Professor Stuart Sechrest © 1991 Christopher A. Warack All Rights Reserved.

Abstract

Specification of the timing properties of real-time systems is a fundamental part of their requirements. Analyzing the timing properties of the system's design and implementation is an important issue for the system developer. Timing analysis is necessary to determine the validity of a design or implementation in respect to the real-time specification.

Using timing schema and PERT networks, Ada program timing behavior can be analyzed. The use of PERT networks is simple but restricted to single processor systems. Replacing the PERT networks with a communicating real-time state machines model allows the analysis of Ada programs on multi-processor systems.

The technique is developed with examples and applied to a Macintosh IIsi programming environment. A foundation is laid for measuring how good a timing analysis prediction fits the implementation.

Acknowledgement

In the short period of my graduate studies I was aided by several friends and associates. I cannot thank them all appropriately, but I would like to single out some of those whose help was critical to this work.

Foremost, I must thank my advisor, Professor Kang Shin. His help and patience has greatly broadened my understanding and appreciation of realtime systems. Secondly, I appreciate the efforts of the other members of my committee, Professors Stuart Sechrest and C.V. Ravishankar.

I would also like to thank Professor Alan Shaw whose work inspired this effort and who took time to answer my questions about his techniques. Major David Umphress at the Air Force Institute of Technology and William S. Salyers of Rational spent valuable portions of their time in getting me access to a Rational computer system. Mike Bonamassa of Rational lent me key material and advice on working with DIANA and tools on the Rational machine.

Finally, I cannot show enough appreciation for the patience, support, and love of my family. My wife, Karen, picked up the slack around the house after a full day of work so I could continue working into the evening. My twoyear-old daughter, Kristina, made sure I did not get completely lost in my work by coming to the office door and asking Daddy for "hugs." They listened, helped, and kept a smile on my face.

iii

Table of Contents

Abstract	ü
Acknowledgement	iii
Introduction	1
Motivation	1
Objectives	3
Organization	4
Execution Time Prediction and Scheduling	6
Introduction	6
Traditional Execution Time Prediction Methods	7
Schema Based Timing Analysis	10
Scheduling Analysis	13
Discussion	15
Timing Schema and Events	16
Introduction	
DIANA Representation of Ada Programs	16
Timing Analysis Transformation Algorithm	20
Timing Schema for DIANA Objects	21
Event Structure for DIANA Objects	
Assertions	
Compiler Analysis	
Execution Time Prediction Algorithm	

Discussion35
Concurrency Model
Introduction
Ada Concurrency Model38
PERT Networks
Communicating Real-Time State Machines43
CRSM and Ada Tasking Structures44
Integrating Schema Analysis and CRSM Construction
Discussion46
Experiments48
Introduction48
Setup of the Experiments
Experimental Results52
Interpretation52
Discussion and Future Research56
Appendices
A. Timing Primitives for Mac IIsi and Meridian Ada60
B. Test Program Source Code63
C. Selected DIANA Representations of Test Programs
D. Experiment Output104
Bibliography

Chapter I

Introduction

Motivation

A gap often exists between stating the requirements for a real-time system and determining if a design matches those requirements. Timing requirements suffer much from lack of adequate attention during the design process since techniques to analyze and abstract timing characteristics are difficult to find or do not trace forward to the implementation. Since the timing characteristics are among the most critical in real-time systems, the ability to track these requirements throughout the development life-cycle is crucial.

The simplest development life-cycle model is the waterfall model. In this model, each activity — specification, design, implementation, and test — occurs sequentially after the completion of the preceding phase. In practice, the waterfall model creates artificial bottlenecks and places unrealistic constraints on the project. More practical developments, however, still conduct the same four basic activities [1], [2]. In them, however, the

activities may occur in different order and repetitiously. Furthermore, different pieces of the project may exist in different activity phases.

The model for software maintenance is quite similar. The fielded system has completed all of the development stages. New parts and modified parts may exist in various stages of completion, however. These additions and modifications are certainly part of the system and cannot be developed completely independent from the fielded system. The key difference between software development and maintenance is that system design and implementation decisions may not be available to the maintainers. While doing the same type of thing, maintainers often have less information outside that embedded in the system.

For an analysis technique to be applicable to the entire project, then, it must be consistent across development activities. The technique must be compatible with analyzing the system's parts existing simultaneously in different stages of development. This requires the technique to apply during design as well as during and after implementation. The results must combine into project-wide results, and these results must relate to the requirements specified for the system.

Typically, real-time systems have finite worst-case response time and workload requirements for specified scenarios. Therefore, a system timing analysis tool must generate predicted response time and workload measures for the system through the various activities of the development life-cycle. Only with this type of support can system developers track and focus on timing requirements during system development and maintenance.

Objectives

This thesis describes a technique for analyzing execution times of Ada programs. The primary objective for development of the analysis technique is that it can apply consistently throughout the development life cycle and answer questions about response times and workload scenarios. Pyster points out that the "hardest parts of developing software are specifying and validating requirements and design" [3]. The primary objective of this thesis focuses on validating the timing properties of the design as well as the implementation. Secondary objectives include:

- Applicability of the technique to different development and target environments with parametric differences only,
- Limited restrictions on design methodology or implementation style, and
- Few, if any, restrictions on applicability to legitimate Ada programs.

The technique is based on source code timing schema [4-6]. The results of applying the schema is used to construct a dependency graph of events connected by code segments where the code segments are represented by their execution times. This graph can be manipulated to determine the worst-case path between two events. The resulting length of the path is the response time. Simultaneous solution for a scenario of events can determine the worst-case processor utilization necessary in a specified period of time, thus determining workload.

Organization

The thesis is organized as follows. Chapters II through IV develop the schema-based execution time analysis algorithm and concurrency extensions. Chapter V presents and summarizes experimental results.

Chapter II develops the background of execution time prediction and scheduling analysis. Existing prediction techniques are evaluated with the criteria described in the objective. Scheduling analysis relies on *a priori* knowledge of task time behavior. The tasks used in the scheduling literature are not necessarily the same as Ada tasks. Thus, a context is developed that connects traditional scheduling analysis and Ada timing analysis. This background sets the stage for development of the analysis technique.

The execution time prediction method develops in Chapter III. Timing schema are defined for a DIANA [7] representation of Ada programs. The schema is defined in terms of primitives that partition language primitive constructs (declarations, statements and expressions) into shared and branch-distinct portions. Furthermore, the schema defines where context switches may occur; these points are defined as events. Ccmpiler analysis generates execution time bounds for each primitive. These steps provide the data for an algorithm to transform DIANA trees into the analysis graph. This graph has events for nodes and code sequences weighted by their execution time bounds for edges.

Chapter IV further develops the concurrency issues involved. It defines the Ada concurrency model. Some limitations on real-time programs naturally fall out of the application of the timing analysis technique to Ada. The next step is to manipulate the analysis graph to generate response time

and workload values. Two techniques are described. One uses Communicating Real-Time State Machines (CRSM) [8]. The other uses PERT analysis techniques.

Experiments are designed to show that the calculated times do indeed bound execution times. Furthermore, simple experiments are hand analyzed for worst-case time and compared to the predicted time. Finally, the predicted, actual and hand-analyzed times are compared to provide a qualitative measure of the technique.

Chapter II

Execution Time Prediction and Scheduling

Introduction

Execution time prediction and scheduling analysis are relatively old problems. Yet, they are far from resolved problems. The standard techniques for determining execution time are primitive. This information, though, is critical to performing scheduling analysis. Only when the execution time of a given *task* is known, then scheduling analysis may be able to determine whether all deadlines can be satisfied. This chapter compares existing time prediction techniques to the criteria of life-cycle applicability, portability, and language limitations. It then discusses how the idea of a *task* in scheduling analysis relates to system design and implementation.

Traditione¹y, the execution time of a program is measured using instruction analysis of the underlying object code or through testing the actual execution time of the implementation. More advanced techniques are developed in [4, 9-14]. These are not used in practice, however, for one reason or another. By comparing these methods to some criteria defining the needs

of real-time Ada program development, motivation develops for a technique derived from Shaw's timing schema [4].

In scheduling analysis *tasks* (italicized for distinction) are commonly defined as a tuple of an arrival time, a period or deadline, and a maximum execution time. This is an abstract notion that relates well to the context of many real-time systems — an event occurs, a *task* is generated to react to it and must complete prior to a deadline; or, an activity must occur periodically and complete prior to the end of the period. This notion of a *task* is very different from the Ada notion of a task. An Ada task is a program construct which exhibits concurrency. These distinct ideas of tasks are resolved by relating a *task* to Ada programming constructs.

Traditional Execution Time Prediction Methods

Two sections in Knuth [15] discuss "Analysis of an Algorithm" and "O-Notation." While asymptotic analysis is useful in making wide distinctions in efficiency, it does not relate directly to time. Deadlines are stated in microseconds or milliseconds, not in O-Notation. Two practical methods are widely used. Knuth also discusses the first of these, the hand analysis of machine object code. He combines this with asymptotic analysis to generate execution time predictions. In general, common sense and logic are used to derive meaningful information from the object code. The other widely-used method is benchmarking or test case monitoring.

The most obvious drawback of these techniques to life-cycle analysis of execution time is that they require object code exist before operating. Thus, they are of limited use during implementation and only fully useful after its completion. Hand analysis is also prone to be highly complex. Knuth's

example [15, pp. 164 - 169] takes an 83 line assembly program and reduces it to a linear equation with six independent variables. To accomplish this required an application of Kirchoff's law to an earlier set of 15 variables and significant application knowledge. In large real-time programs, this complexity is overwhelming. On the other hand, testing suffers from lack of rigor. Unless careful analysis shows that the test cases generate a relationship to the worst case execution time for the code, then these results are not necessarily legitimate bounds. Possibly, the derived times may bound the execution time *most of the time*. This qualification, though, is not quantifiable and the consequences of failure may be too severe to rely on it. Thus, we learn from these techniques a need for analytic simplicity and logical rigor. Structured programming in higher-level languages was developed to simplify the logical complexity of unstructured and assembler code. Several researchers have turned toward analysis of the source code for timing information to benefit from its reduced complexity.

Mok's annotation technique [11] automates the hand analysis process described by Knuth. Like Knuth's technique it is limited to the implementation and post-implementation phases. The developer annotates the program. The annotated program is fed to a set of timing analysis tools and to a special compiler. The compiler is modified only in that it adds labels to various assembly instructions in the code generation phase. These labels do not affect the final object code that the assembler stage generates. Besides feeding the assembler stage, the annotated assembly code is also fed to the timing analysis tool. Using the labels, the annotated source code and assembly code are merged. The developer then works interactively with the timing tools to generate timing information about the program.

Many of the more advanced techniques, including Haase's guarded commands and PARCs [9], Halang's extensions to the PEARL programming language [10], Puschner and Koza's MARS-C language [12], and Kenney and Lin's FLEX [13], are based on theoretical languages or language extensions that are not used in practice. Thus, they are not portable as defined nor do they use Ada. Many of these extensions could be translated to Ada; however, Ada supersets are disallowed in Department of Defense projects and are discouraged in general. This raises the restriction that the technique not require language extensions or particular code generation behavior beyond that stated in the language definition. It is important to note that annotations like those used by Mok satisfy this restriction. Although they require support in the compiler, they do not affect the language definition.

Glicker and Hosch describe a system that uses symbolic execution to model the behavior of Ada programs [14]. It first determines the best and worst case threads of execution through the Ada task system given a set of preconditions. These threads are then measured directly using the target architecture. This method shows some promise although it can only apply to completed programs. Its success hinges on the adequacy of the symbolic execution stage. Tracing all threads through a program is naturally an $O(e^x)$ time activity. Good branch-and-bound heuristics must be applied to keep the process tractable.

Several of these techniques share common features. The most prevalent of these is analysis of the programming language constructs themselves to reason about the timing behavior of the program. This concept is distilled by Shaw in his timing schema approach. Shaw's work is done in C. Ada has several language constructs that make timing analysis easier and several

more complex constructs than the simple statements and expressions allowed by C.

In summary, existing techniques are generally not applicable to large realtime Ada projects. Furthermore, they are not applicable throughout the development process. Mok's annotations could be extended to Ada's sequential language constructs. Its biggest short-coming is reliance on compiler support. It also raises some configuration management hurdles in ensuring that the data under analysis comes from the current code baseline. Jumping these hurdles is straight-forward with careful process management. Shaw's timing schema grant the developer less flexibility than Mok's for timing analysis, but do not rely on compiler support. Both techniques must be extended to handle Ada tasking constructs. This thesis' technique develops Shaw's timing schema, adding tasking support, while allowing some compiler-independent annotations or *assertions*.

Schema Based Timing Analysis

Timing schema are based on Hoare logic [16]. Hoare logic uses the notation $\{P\}S\{Q\}$ to mean that given the conditions P immediately prior to execution of S results in condition Q upon completion of execution. An inherent assumption is that S does, in fact, complete. The change in time from executing statement S can be described with Hoare logic as follows $\{rt = x\}S\{rt = x + t(S)\}$, where rt represents the time, x represents the value of rt immediately prior to execution of S and t is a function which returns the execution time of S. The first application of Hoare logic like this is in a paper by Mary Shaw. [17]

The function t has as its domain the set of all possible programs and as its range the non-negative real numbers. A definition of t can consist of a set of axioms {(S,t)} relating all programs to their execution time. Since there are infinite programs, this approach cannot always work. However, t can be described with a finite set of rules for generating the infinite set of relations above. This set of rules is a schema.

The above is flawed in that t in the rule set must be a single value to define a well-formed function. For a single execution of a program, it will indeed take a specific time value to execute. In fact, the execution time of a given program may vary from one execution to another. There exist many reasons for this including differences in inputs, differences in machine state at the start of execution, inconsistencies in the machine clocking mechanism and differences in the machine-language instantiation of the program. Further discussion of these variabilities will occur in the section on compiler analysis in Chapter 3. Let t'(S,x) be the function which returns the single value for the execution time of S as indexed by $x \in \{all \text{ possible execution} \}$ conditions). A more correct model of t, then, may be a random variable; thus t is a mapping from programs to random variables. It is adequate, however, to model t as a closed interval $T = [t_{min}, t_{max}]$ where $P[t'(S,x) \in T] = 1$ for all x. Therefore, timing analysis of a program S consists of computing t(S). This is done using a schema which generates a rule computing a closed time interval, T, for S.

Arithmetic and logical operations on values of T use a form of interval arithmetic. Additive operations are simply applied component-wise. Multiplicative operations are distributed, T op $x = [t_{min}$ op x, t_{max} op x]. Relational operations are applied to the t_{max} component first since the worst-

case is the more important in real-time analysis. If the t_{max} components are equal, then comparison of the t_{min} components determines the result.

The schema are defined for the grammatical elements of a language. For instance, the schema for an if-statement, T[if-statement], might be:

 $\begin{aligned} \mathcal{I}[\texttt{if-statement}] &= [10,15] + \mathcal{I}[\texttt{boolean-exp}] + [\texttt{min}(([4,6] + \mathcal{I}[\texttt{then-part}]), \\ & ([2,4] + \mathcal{I}[\texttt{else-part}])), \\ & \texttt{max}(([4,6] + \mathcal{I}[\texttt{then-part}]), ([2,4] + \mathcal{I}[\texttt{else-part}]))] \\ & \texttt{where if-statement ::= if boolean-exp then then-part else else-part;} \end{aligned}$

The schema rule's definition consists of constant parts like [10,15] and recursive invocation of other schema rules. These constant parts represent some basic computation to provide the language behavior for that construct. The logical basis of the above rule is that the system computes the boolean expression (boolean-exp) and spends time branching on the result ([10,15]). The branch with the bigger execution time is the worst case choice and the lesser execution time is the best case choice. The branch execution time consists of the time spent executing the statements in the branch (then-part or else-part) and time spent rejoining the main execution stream ([4,6] or [2,4]).

Primitive times like the constant parts described above are dependent on the underlying system consisting of the hardware, operating system, and compiler. The exact value of the primitives will differ from one underlying system to another, but it is always present in the schema since its existence derives from some computational need in the language definition. It may be possible for a particular system to compute a particular primitive in time $[0,\varepsilon]$, for a very small $\varepsilon > 0$, using special hardware or subsuming it in other primitives.

Eventually, any given program will reduce through the schema rules to a sum of primitive times. This is akin to parsing the language where nonterminals in the grammar are similar to schema rules and terminals are similar to primitive times. Computing the sum gives the resulting execution time of the code sequence. One inherent assumption throughout this discussion has been that the program has a single *thread* of execution. With multiple threads, timing schema analysis must be applied to each thread. The relationship between the threads must be addressed with other mechanisms. These mechanisms will be discussed in the next two chapters.

Scheduling Analysis

Scheduling is one of the main problems in the area of real-time systems. The problem is to determine if a given set of tasks can meet all of their deadlines on a given set of processors. Tasks in the sense of scheduling are described as a triple $\{a, c, d\}$ where a is the arrival time, c is the execution time required to complete the task, and d is the deadline for the task [18]. Knowing the execution time of a task, then, is critical to conducting scheduling analysis.

This notion of a task, however, is abstract. An Ada task, on the other hand, is a concrete programming construct. The two concepts do not necessarily relate directly. Consider an event-driven system. The response of the system to a certain input event is a task for scheduling purposes. Its arrival time is the time of the event. Its deadline is the response time specified for the system (derived from physical requirements or allocation of other timing requirements). The final component is the execution time. While the first two components are commonly defined in the system requirements or description, the execution time results from design and implementation. Timing analysis is the technique to determine the execution time. During the requirements and early design phase, the software analyst should determine the events, deadlines, and event arrival scenarios the system will handle. Scheduling analysis is used to determine if the system can satisfy its constraints. Thus, execution times must be predicted as early in the design as possible. As design progresses and implementation begins, the system model is refined. Scheduling analysis continues to determine if the current model is viable. Thus, execution timing analysis must continue and hopefully improve through the process. During validation, scheduling analysis with inputs from timing analysis is used to determine whether or not the system as implemented can satisfy its constraints in all cases. Testing is not satisfactory in many cases since it may not test the worst case conditions that may be encountered.

Scheduling analysis, however, depends on the scheduling strategy implemented by the system. A rate monotonic system can be implemented using Ada tasking and priorities. Ada, however, is currently prone to priority inversion [19]. Many scheduling approaches are available in any programming language by implementing a scheduler as part of the system. Cyclic executives are also popular with real-time developers [20].

In summary, it is important to keep distinct the concept of scheduling *tasks* and the Ada task constructs. While it is possible, Ada tasks do not map well to current scheduling strategies. Scheduling strategies can be built into a system, however, and Ada tasks used within the implementation of that system separate from scheduling policies.

Discussion

Timing Analysis using Shaw's schema techniques and Ada satisfies the stated objectives. When used with scheduling analysis, timing analysis can be used to show that a system satisfies its real-time requirements. Regardless, timing analysis can characterize program timing behavior to give system developers information for design, implementation and verification decisions.

Several existing timing analysis techniques only apply to completely implemented systems. The timing schema approach outlined above can be used with an Ada program design language (PDL) to provide timing analysis throughout the development and with varying stages of system completion. This lets the developer identify and track or correct problems early when they are cheapest to fix.

Chapter III

Timing Schema and Events

Introduction

The first step in timing analysis is transforming the Ada programs into timing graphs. The transformation is based on a DIANA representation of these programs. A schema rule is defined for each type of DIANA node. The resulting timing graph may be as simple as a single edge, representing a simple sequential program. It may also be very complex containing several nodes and branching alternatives. The generation of these graphs are discussed. The analyst has some control over the graphs through the use of assertions.

DIANA Representation of Ada Programs

DIANA is an abstract data type for representing Ada programs [7]. A DIANA object is mathematically modelled as an attributed tree. The tree represents a normalized form of a corresponding Ada program. It also guarantees that a given Ada object has only one defining occurrence; and the

defining occurrence is an attribute of the other occurrences. Using DIANA takes care of the complicated Ada parsing and static semantic analysis. The tree model is easily manipulated.

A given node of the DIANA tree is defined in terms of the attributes it has. A node has a structural arity in the set {0,1,2,3,n} and has zero or more lexical, semantic, and code attributes. Additional attributes may exist as needed by an application; however, these are not standard and cannot be relied on. The structural arity denotes the branching of each node to other nodes. The other attributes may provide numeric or textual information or be semantically related nodes. By sharing identical nodes, the tree becomes a directed acyclic graph (DAG). The DAG model is identical to the tree model except that it allows replication to be eliminated.

For example, the simple program below simulates rolling x n-sided dice where x and n are supplied by the caller. This converts to the DIANA DAG described following it. The format for a DAG node is name¹ : node_type [attributes]. Attributes are juxtaposed pairs of the form attribute name attribute value. Multiple attributes are separated by semicolons.

```
function ROLL_DICE (NUM_SIDES, NUM_DICE : in INTEGER)
return INTEGER is
subtype DIE_RANGE is INTEGER 1 .. NUM_SIDES;
A_DIE : DIE_RANGE;
TOTAL : INTEGER := 0;
begin
for A_ROLL in 1 .. NUM_DICE loop
A_DIE := INTEGER(RANDOM * NUM_SIDES) + 1;
TOTAL := TOTAL + A_DIE;
end loop;
return TOTAL;
end ROLL DICE;
```

Figure 1: Sample Ada Program

¹ Nodes with names like PDx are part of the Ada package "Standard" provided as part of the compiler environment. These names are used consistently with the example in [7].

 			-	
A1	:	comp unit	[as_pragma_s ^A2;
				as_context ^A3;
				as unit body ^A4]
A2	:	pragma s	Ι	as_list < >]
A3	:	context	[as_list < > }
A4	:	subprogram body	Ĩ	as_designator ^A5;
			-	as header ^A6;
				as_block_stub ^A7]
A5	:	function id	[<pre>lx_symrep "ROLL_DICE";</pre>
		—	•	sm spec ^A6
				sm body ^A7
				sm location void]
A6	:	function	ſ	as param s ^A8
			-	as name ^A13]
A7	:	block	ſ	as item s ^A14;
			•	as stm s ^A29:
				as alternative s ^A54 1
ЪŔ	•	naram s	ſ	as list $< ^A9 > 1$
<u>д</u> 9		in	r	as id s A10 :
	•	±	L	as name ^Al3:
				as exp void void 1
A10	•	id s	ſ	as list $< ^{A11} ^{A12} > 1$
A11	•	in id	r	lx symrep "NUM SIDES":
MI I	•	111_10	L	sm init exp void:
				sm obj tvpe ^PD9 1
12		in id	r	lx symrep "NUM DICE":
	•	111_14	L	sm init exp void:
				sm obj type ^PD9 1
גומ	•	used name id^2	r	lx symrep "INTEGER":
AI J	•	dbed_name_id	L	sm defn ^PD8 1
14 ه	•	itom a	T	as list < 150 , 16 17 > 1
A15	•	subtype	r	as id 118
ALD	•	subcype	L	as constrained ^119 1
A 16		war	r	as id s 123
AT0	•	Val	ι	as type spec 25
				as_cype_spec AZS,
N 1 7	-			as_object_der vord j
AI/	•	Var	ĩ	as_{10} s $Azo;$
				as_cype_spec AIS;
N10		aubturna id	T	AS_ODJECC_GEI AZO J
AIO	÷	supcype_id	L	m tune sneg ^119 1
N10		constrained	r	sm_cype_spec Ary j
ALA	•	constrained	L	as_name AIJ;
				$as_constraint Azo;$
				$sm_cype_scruce Arg;$
				sm_base_cype PD5;
N 20		****	r	$sm_constraint A20 j$
AZU	:	range	ι	ap_exp1 A21;
		numeria literal		$as_{as_{as_{as_{as_{as_{as_{as_{as_{as_{$
AZ I	:	numeric_literal	L	$\frac{1}{2} \frac{1}{2} \frac{1}$
				$sm_exp_cype_rus;$
		used object id	,	SM_VALUE I J
n 22	÷	used_onlect_1d	ι	am own tano von stres :
				om dofn ^%11 1
		:	,	$a_{1}a_{1}a_{1}a_{1}a_{1}a_{1}a_{1}a_{1}$
AZJ		TO B	1	as 113L N A24 /

Note that this node is heavily reused in the structural DAG. This is not surprising since it represents the type integer.

 A24	:	var id]	lx symrep "A DIE";
			•	sm obj type ^A19;
				sm address void;
				sm obj_def void]
A25	:	used_name_id	[<pre>lx_symrep "DIE_RANGE";</pre>
		-	-	sm_defn ^A18]
A26	:	id_s	Ε	as_list < ^A27 >]
A27	:	var_id	[<pre>lx_symrep "TOTAL";</pre>
		—		<pre>sm_obj_type ^PD9;</pre>
				sm_address void;
				sm_obj_def ^A28]
A28	:	numeric_literal]	<pre>lx_numrep "0";</pre>
				<pre>sm_exp_type ^PD9;</pre>
				sm_value 0]
A29	:	stm s	[as list < ^A30 ^A53 >]
A30	:	loop	ī	as iteration ^A31;
	-	<u>-</u>	•	as stm s ^A36 1
A31	:	for	1	as id ^A32;
	-		•	as dscrt range ^A33]
A32	:	iteration id	1	lx svmrep "A ROLL";
			•	sm obj type ^PD9 1
A33	:	constrained	1	as name ^A13;
_			•	as constraint ^A34;
				sm tvpe struct ^A33;
				sm base type ^PD9;
				sm constraint ^A34]
A34	:	range	ſ	as expl ^A21;
	•	141.30	•	as $exp2^{A35}$ 1
A35	:	used object id	r	ly symrep "NUM DICE":
1100	•		L	sm exp type ^PD9:
				sm defn A12 1
A36	:	stm s	ſ	as list < $^{A37} ^{A49} > 1$
A37	:	assign	'n	as name ^A38;
	•		•	as exp ^A39 1
A38	:	used object id	ſ	lx symrep "A DIE":
*** -	•		Ľ	$m_{sm} exp type ^A19:$
				$sm_defn ^{A24}$ 1
A39	•	function call	ſ	as name ^A40:
****	•	Lungerou-ours	L	as naram assoc s A41
A40	:	used bltn op	ſ	lx symrep "+":
****	•			sm operator BINARY PLUS 1
A 41	•	naram assoc s	r	as ligt < $A42 A21 > 1$
A42	•	conversion	ľ	as name 13 :
****	•	CONVEL BION	L	as $\alpha n \Lambda A 3$ 1
A 43	•	function call	ſ	as name $^{\Lambda44}$:
n45	•	runetion_ourr	ĩ	as naram agon s^A45
D 4 4	•	used bltn op	г	ly sumron "*".
U.J. 3	•	npea_prow_of	r	am operator MIII.TTPI.Y]
A45	•	neram aggor g	r	as list < $^A46 ^A22 > 1$
A46	•	function call	r	as name A47 :
****	•	runouzon_ouzz	Ľ	as naram assoc s A48 1
▶47	÷	used object id	r	ly gumren "RANDOM":
N3,	•	usea_objecc_ia	L	am evo tune
				$am defn 1^3$
> 48	•	naram aggor g	r	$Sim_defin \dots $
1140	•			

³ This is an external function with nodes outside the immediate program. These are elided for conciseness.

			20	
A49) :	assign	[as_name ^A50; as_exp ^A51]	
A50) :	used_object_id	<pre>[lx_symrep "TOTAL"; sm_exp_type ^PD9; sm_defn_^A27]</pre>	
A51	. :	function_call	[as_name ^A40; as param assoc s ^A52]	
A52	: :	param assoc s	[as list < ^A50 ^A38 >]	
A53	3 :	return	[as exp void ^A50]	
A54	:	alternative_s	[as list < >]	

Figure 2: DIANA Representation of the sample program



Figure 3: DAG of structural DIANA nodes for Sample Program

Timing Analysis Transformation Algorithm

The basic algorithm is to transform the DIANA object representing an Ada program into a graph. The edges of the graph represent sequential portions of the Ada program. The vertices represent potential context switches. The graph is built by traversing the tree depth-first or "bottom-up." As each node is traversed a subgraph is created based on the type of node and the subgraphs of its children. A simple implementation is to apply the schema function to the root node and using the recursive nature of the schema to traverse the tree. The traversal of a node is dependent on its type and structural attributes only. Its schema, however, may use semantic information in computing bounds and values.

The timing graph created by this process consist of edges weighted with execution times and vertices to connect edges. Multiple edges leaving a single vertex represent branching dependent on task synchronization. When constructed into a system network, only one branch in an instance will be utilized; the others are discarded.

A delay edge denotes a constraint that a certain time must pass between two vertices before execution can continue. A context switching node is so marked where a context switch may occur. This is done solely for the purposes of calculating the potential number of context switches in the resulting system model.

Timing Schema for DIANA Objects

The 170 different DIANA node types are listed below. Each entry includes the structural and key semantic attributes as well as the schema for computing the worst and best case time bounds for that node type. Several auxiliary functions are used to simplify the schema. These include:

- Store(Node): Determines the proper primitive time to store a value in an object by examining the type of the object.
- Access(Node): Determines the proper primitive time to access a value in an object by examining the type of the object.

- Init(Node): Determines the proper primitive time to initialize a value in an object by examining the type of the object.
- Save: This function updates the library of computed timing graphs with the graph computed for some subprogram, task, package, or generic declaration. These graphs are used by the Insert function or by the analyst in constructing a system network as described in the next chapter. Save "returns" the value [0,0]; that is, in storing the graph designated, this graph is not included in the computed time of the declaration block in which it occurs. This follows since it is not executed at time of declaration. Some elaboration time may be included in the schema in addition to the save, however. This does, indeed, execute during elaboration of the execution block.
- Stop: Halt computation within an enclosing stm_s type node. This occurs when an unconditional change in control flow is encountered in a sequence (i.e., return, goto, or unconditional exit).
- Abort: Ignore this path (stm_s node) since it contains a raise or abort statement. Currently raise and abort are restricted to use with error conditions. Error conditions are not analyzed by the technique at this time.
- Insert(Name): Insert the graph for a subprogram at this point in the current graph. Look for a recursion assertion if it is the same subprogram.
- Node(Name): End the current edge. Create a vertex. Start a new edge(s). With rendezvouses, the node will be linked to a corresponding node in another task, so one edge will be missing.
- Delay(Duration): Create a pair of nodes with a delay edge between them of the duration given.
- Activate(Task[s]): Create a context-switching node at this point preceded by the timing primitive P(activation) and attach Task[s] (as well as continuing the current thread).
- Queue-Activate(Task[s]): Like Activate except that the node is inserted after completing all processing of the current declarative block.
- ConstraintCheck(Node): Determine the proper primitive time for a constraint check on the type of the given node.
- Print(String): Print the given string on the analyzer's error output stream.
- Range(DSCRT_RANGE): Determines the lowest and highest possible values of the range.

Node Type	Arity	Schema
abort	1	Abort abort statements are treated as errors at this point and are not
		candidates for a min or max path.
accept	3	P(accept) + Node("start "&as_name); Node("Begin"&as_name) +
		P(rendezvous) + T(as_stm_s) + P(accept_end) + Node("end "&as_name)
access	1	T(as_constrained)
address	2	[0,0] representation clause affects compilation only
aggregate	n	(\mathbf{n})
		$\left(\sum T(as list[i])\right) + Store(sm exp type)$
		$\begin{pmatrix} 2 \\ i=1 \end{pmatrix}$
alignment	2	[0.0] representation clause affects compilation only
all	1	[0,0] · representation clause anects complication only
allocator	1	T(as eve constrained) +
anocaon	1	if sm even type = task spec
		then Activate(task)
		else Store(sm_exp_type) + P(allocate_mem(Size(sm_exp_type))
		Note: Records and arrays with task components need a hybrid of the if-
		statement above to apply.
alternative	2	subsumed in schema for case node
		ignored as attribute of exception part of blocks
alternative_s	n	subsumed in schema for case node
		ignored as attribute of exception part of blocks
and_then	0	P(and_then)
argument_id	0	[0,0] identifier "symbol table" entry
array	2	T(as_dscrt_range_s) + T(as_constrained)
assign	2	T(as_name) + T(as_exp) + Store(as_name) + ConstraintCheck(as_name)
		+ if as_exp = used_object_id then Access(as_exp.sm_exp_type) else [0,0]
assoc	2	T(as_actual)
attr_id	0	[0,0] identifier "symbol table" entry
attribute	2	T(as_name) + P("attr_" & as_id.sm_defn.lx_symrep)
		attribute execution times are pre-defined
attribute_call	2	T(as_exp) + T(as_name) as_name is a node of type attribute
binary	3	$[T(as_exp1), T(as_exp1) + T(as_exp2)] + T(as_binary_op)$
plock	3	$T(as_item_s) + T(as_stm_s)$
	L	as_alternative_s represents exception handlers
DOX	<u> </u>	[0,0] generic subprogram formal option
case	2	$1(as_exp) + P(case) + [min(choices), max(choices)]$
		$cnoices = \{1(as_anernative_s.as_nsu[n].as_sun_s\}: 1 \le n \le as_nsu \}$
choice a		- as_anernative_s.as_nsunj.as_choice_s must be stauc and is ignored
choice_s	"	$\Sigma T (ac list[i])$
		$\sum_{j=1}^{n} \frac{1}{as_{j}} \frac{1}{s_{j}} \frac{1}{s_{j}}$
code	2	$T_{a} = T_{a}$
		- execution time bounds must be specified with a time assertion
comp id	0	[0.0] identifier "symbol table" entry
comp rep	3	[0.0] representation clause affects compilation only
comp rep s	n	[0.0] representation clause affects compilation only
comp unit	3	T(as unit body)
comp_	Ű	as context and as pragma s set up environment only
compilation	n	(map T as list) = T(hd(as list)):(map T tl(as list))
-		list of all compilation units in the system
cond_clause	2	subsumed in schema for "if" node
cond_entry	2	P(cond_entry) + Node1 + Delay(e) + T(as_stm_s2) + Node2;
· ·		Delay(e) prejudices choice
		Node1 + T(as_stm_s1) + Node2; first stm is entry_call
const_id	0	[0,0] identifier "symbol table" entry
constant	3	T(as_type_spec) + T(as_object_def)
constrained	2	T(as_constraint)
context	n	[0,0] set's up environment, no cost

conversion	2	case as_name.sm_defn.sm_type_spec, as_exp.sm_type_spec
		real to int: P(convert_float2int) or P(convert_fixed2int)
		int to real: P(convert_int2float or P(convert_int2fixed)
		real to real: P(convert_fixed2float) or P(convert_float2fixed)
		derived : P(convert derived)
		array: P(convert array)
		others: [0.0] not changing base type
		if subtype
		then Constraint Check(am evo type) else [0,0]
		+ T(as exp)
decla	n	n
ucci_5		Trag ligtfil) + Init(ag ligtfil)
		$\sum_{i=1}^{n} \frac{1}{2} \left[\frac{1}{2} $
def eben		$\frac{1-1}{1-1}$
del_char	U	[0,0], never caned since enum_interal_s is always [0,0]
1.0		$Access = P(num_access)$
der_op	0	[0,0] operator "symbol table" entry
deferred_constant	2	[0,0] any cost will be incurred with full declaration
delay	1	$T(as_exp) + P(delay)$
		+ Delay((sm_value or max(subtype range)) + P(delay_cap))
derived	1	T(as_constrained)
dscrmt_aggregate	n	n
		$\sum T(as_list[i])$
		i=1
dscrmt_id	0	[0,0] identifier "symbol table" entry
dscrmt_var	3	T(as_object_def)
dscrmt var s	n	n
		$\overline{\mathbf{V}}$ TY as list[i])
		i=1
decrt range a	n	n
userv_range_s	-	TYpe list[i])
		$\sum_{i=1}^{n} 1(as_{i}s_{i}s_{i}s_{i})$
A		
entry	2	$1(as_ascrt_range_vold) + 1(as_param_s)$
entry_call	z	T(as_name) + T(sm_normalized_param_s) + P(queue_entry)
		+ Node("start "&as_name); Node("end "&as_name) + $[0,0]$
entry_id	0	[0,0] identifier "symbol table" entry
enum_id	0	[0,0]; never called since enum_literal_s is always [0,0]
enum_literal_s	n	[0,0] list of def_char and enum_id; each static so [0,0] elaboration time
exception	2	[0,0] declares exception names
exception_id	0	[0,0] identifier "symbol table" entry
exit	2	if as exp void = void then Stop* else T(as exp void)
	_	computes condition, but doesn't affect number of loop iterations
		- *a branch containing unconditional exit can only be executed once
		so normally connect he worst case branch 4
	-	so normany cannot be worst case branch
exp_s	<u>п</u>	$\mathbf{\Sigma} \mathbf{m} = 1 + 1 + 1$
		1=1
I TIYON		
lixed	2	T(as_range_void) as_exp is static
float	2 2	T(as_range_void) as_exp is static T(as_range_void) as_exp is static
float for	2 2 2	T(as_range_void) as_exp is static T(as_range_void) as_exp is static T(as_dscrt_range)
float for formal_dscrt	2 2 2 0	T(as_range_void) as_exp is static T(as_range_void) as_exp is static T(as_dscrt_range) [0,0] place holder for generic type parameters
float for formal_dscrt formal_fixed	2 2 2 0 0	T(as_range_void) as_exp is static T(as_range_void) as_exp is static T(as_dscrt_range) [0,0] place holder for generic type parameters [0,0] place holder for generic type parameters
float for formal_dscrt formal_fixed formal_float	2 2 2 0 0 0	T(as_range_void) as_exp is static T(as_range_void) as_exp is static T(as_dscrt_range) [0,0] place holder for generic type parameters
float for formal_dscrt formal_fixed formal_float formal_integer	2 2 0 0 0 0	T(as_range_void) as_exp is static T(as_range_void) as_exp is static T(as_dscrt_range) [0,0] place holder for generic type parameters

⁴ Can be best case, however. Furthermore, unusual conditions such as an exit branch which is much more computationally intensive than other branches through the loop or a loop which only executes a small number of times may be a worst case. A more thorough analysis may determine this.

function_call	2	if as_name is used_bltn_id or used_bltn_op then check sm_value for static expression and treat as folded constant if one exists, otherwise P(bltn) + T(sm_normalized_param_s)
		else P(function_call) + T(sm_normalized_param_s) + Insert(as_name) + P(function_end)
		sm_normalized_param_s includes default params
function_id	0	[0,0] identifier "symbol table" entry
generic	3	save id(as_id) = T(as_generic_header)
	:	as_generic_param_s is used to define quasi-primitives to be replaced with actual times when instantiated.
generic_assoc_s	n	$\sum_{i=1}^{n} T(as_list[i])$
generic_id	0	[0,0] identifier "symbol table" entry
generic_param_s	n	n
		$\sum_{i=1}^{T(as_list[i])}$
goto	1	Print("Warning: Unstructured statement <goto> cannot be analyzed")</goto>
		 Stop if a forward goto, this will compute correctly; if it causes a loop, then the analysis is bad.
id_s	n	[0,0];
if	1	<i>c</i> / n
		choices = $\left\{ \left(\sum_{i=1}^{n} T(as_{i}].as_{exp_void}) + P(else) \right) + \right\}$
		$T(as_list[n].as_stm_s) : 1 \le n \le as_list \}$
in	3	if as any word + word
	U	then $T(as exp void) + P(default naram)$
in id	0	[0.0] identifier "symbol table" entry
in op	0	P(in)
in_out	3	[0,0] cannot have default parameters
in_out_id	0	[0,0] identifier "symbol table" entry
index	1	[0,0] as name is an uconstrained type.
indexed	2	T(as_name) + T(as_exp_s);
inner_record	n	$\sum_{i=1}^{n} T(as_list[i])$
instantiation	2	T(as_generic_assoc_s)
integer	1	T(as_range);
item_s	n	$\sum_{i=1}^{n} T(as_list(i))$
iteration_id	0	[0,0] identifier "symbol table" entry
label_id	0	[0,0] identifier "symbol table" entry
labeled	2	T(as_stm)
loop	2	if as_iteration = void or as_iteration = while
		then T(as_iteration) + P(loop) + LOOP_ASSERTION * (T(as_stm_s) + T(as_iteration) + P(iter))
		• P(iter) + P(loop_end)
		II no loop assertion then II no nodes in as_stm_s
		else unroll as far as necessary
		else T(as iteration) + P(for loop) +
		Range(as_iteration_as_dscrt_range) * (T(as_stm_s) + P(for_iter))
		- P(for_iter) + P(for_end)
l_private	0	[0,0]
l_private_type_id	0	[0,0] identifier "symbol table" entry
membership	3	T(as_exp) + T(as_type_range) + T(as_membership_op)

name s	n	n
	_	$\overline{\Sigma}$ T(as list(i))
		i=1
named	2	T(as choice s) + T(as exp)
named stm	2	T(as_stm)
named atm id		[0 0] identifier "gymbol table" entry
na default	0	[0,0] - rachance symbol able chary
not in	0	D(not in)
	0	
null_access	<u> </u>	
null_comp	0	
null_stm	0	P(null)
number	2	[0,0] static numeric constant
number_id	0	[0,0] identifier "symbol table" entry
numeric_literal	0	if $sm_value \leq SMALL_VAL$ then $P(small_num_literal_access)$ else $P(num_literal_access)$ where $num_lis_int_float$ or fixed
or else	0	P(or else)
others	0	
out	3	[0,0] cannot have default narameters
out id	0	[0,0] - identifier "gymbal table" entry
vac_ia	2	[0,0] - identifier symbol table entry
package_bouy	4	save $Id(as_id) = T(as_body) = F(package_elaboration) + T(as_block_stub)$
package_dect	4	if as_package_def is rename or instantiation then use other values
package id	0	[0,0] identifier "symbol table" entry
package spec	2	T(as decl s1) + T(as decl s2)
Daram assoc s	n	n
purum_uscoc_s	-	$\tilde{\Sigma}$ Tree light(i) - actual parameter light
		i=1
nerem a	n	
param_s		$\sum_{i=1}^{n} T(x_{i}, y_{i})$
		2 1(as_iisqij)
nerenthesized	1	Tree evo)
		1(do_CAP)
pregme	2	[0.0] - compiler directive may change global parama, but no code gen
pragma	2	[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled of course
pragma	2	[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course.
pragma_id	2 0	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen
pragma_id pragma_s	2 0 <u>n</u>	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen
pragma_id pragma_s private	2 0 <u>n</u> 0	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0]
pragma_id pragma_s private private_type_id	2 0 <u>n</u> 0 0	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry
pragma_id pragma_s private private_type_id proc_id	2 0 0 0 0	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry
pragma_id pragma_s private private_type_id proc_id procedure	2 0 <u>n</u> 0 0 0 1	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry
pragma_id pragma_s private private_type_id proc_id procedure procedure_call	2 0 0 0 0 1 2	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] identifier "symbol table" entry
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call	2 0 <u>n</u> 0 0 0 1 2	[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [3,0] identifier "symbol table" entry [4,0] identifier "symbol table" entry
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call	2 0 <u>n</u> 0 0 0 1 2	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [3,0] identifier "symbol table" entry [4,0] identifier "symbol table" entry [5,0] identifier "symbol table" entry [6,0] identifier "symbol table" entry [7(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified	2 0 0 0 1 2 2	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] identifier "symbol table" entry [1,0] identifier "symbol t
pragma_id pragma_s private_type_id proc_id procedure_ procedure_call qualified raise	$\begin{array}{c} 2\\ 0\\ n\\ 0\\ 0\\ 0\\ 1\\ 2\\ 2\\ 1\\ 1 \end{array}$	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [3,0] identifier "symbol table" entry [4,0] identifier "symbol table" entry [5,0] identifier "symbol table" entry [7(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params [7(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise	$\begin{array}{c} 2\\ 0\\ \underline{n}\\ 0\\ 0\\ 0\\ 1\\ 2\\ 2\\ 1\\ 1\\ 2\\ 2\\ 1\\ 2\\ 2\\ 1\\ 2\\ 1\\ 2\\ 1\\ 2\\ 1\\ 2\\ 2\\ 2\\ 1\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\ 2\\$	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [3,0] identifier "symbol table" entry [4,0] identifier "symbol table" entry [5,0] identifier "symbol table" entry [6,0] identifier "symbol table" entry [7(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params [7(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path.
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range	2 0 <u>n</u> 0 0 1 2 2 1 2	 [0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [1,0] identifier "symbol table" entry [2,0] identifier "symbol table" entry [3,0] identifier "symbol table" entry [4,0] identifier "symbol table" entry [5,0] identifier "symbol table" entry [6,0] identifier "symbol table" entry [7(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params [7(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. [7(as_exp1) + T(as_exp2)
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record	2 0 <u>n</u> 0 0 1 2 2 1 2 1 2 1	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record	2 0 <u>n</u> 0 0 1 2 1 2 1 <u>2</u> <u>1</u> <u>2</u> <u>1</u> <u>2</u> <u>1</u>	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n \sum_T(as_list[i])</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record	2 0 <u>n</u> 0 0 1 2 1 2 1 2 n	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n S_T(as_list[i]) i=1</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep	2 0 <u>n</u> 0 0 1 2 2 1 2 1 3	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n \$\sum_{i=1}^{N} T(as_list[i])\$ i=1 [0,0] representation clause affects compilation only</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record_rep rename	2 0 <u>n</u> 0 0 1 2 1 2 1 3 1	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n \$\sum_{i=1}^{\text{T(as_list[i])}} i=1</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record_rep rename return	2 0 <u>n</u> 0 0 1 2 2 1 2 1 3 1 1	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n S_T(as_list[i]) i=1 [0,0] representation clause affects compilation only [0,0] T(as_exp_void) + Store(function_call) + Stop</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse	2 0 n 0 0 1 2 1 2 1 2 1 3 1 1 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n S_T(as_list[i]) i=1 [0,0] representation clause affects compilation only [0,0] T(as_exp_void) + Store(function_call) + Stop T(as_dscrt_range)</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse select	2 0 n 0 0 1 2 1 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n \sum_T(as_list[i]) i=1 [0,0] representation clause affects compilation only [0,0] T(as_exp_void) + Store(function_call) + Stop T(as_dscrt_range) P(select)</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse select	2 0 n 0 0 1 2 1 2 1 1 2 1 1 2 2 2 1 1 2 2 2 1 2 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n \sum_T(as_list[i]) i=1 [0,0] representation clause affects compilation only [0,0] T(as_exp_void) + Store(function_call) + Stop T(as_dscrt_range) P(select) as_select_clauses_s </pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse select	2 0 n 0 0 1 2 1 2 1 2 1 1 2 2 1 2 2 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n T(as_list[i]) i=1 [0,0] representation clause affects compilation only [0,0] T(as_dscrt_range) P(select) i as_select_clauses_si +</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse select	2 0 n 0 0 1 2 1 2 1 1 2 1 1 2 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 2 2 1 2 2 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n T(as_list[i]) i=1 T(as_list[i]) [0,0] representation clause affects compilation only [0,0] T(as_exp_void) + Store(function_call) + Stop T(as_dscrt_range) P(select) as_select_clauses_s +</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse select	2 0 n 0 0 1 2 1 2 1 1 2 2 1 1 2 2 2 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry T(as_param_s) T(sm_normalized_param_s) + P(procedure_call) + Insert(as_name) + P(procedure_end) sm_normalized_param_s includes default params T(as_exp) as_name only used by compiler Abort exceptions are treated as errors at this point and are not candidates for a min or max path. T(as_exp1) + T(as_exp2) n ∑ T(as_list[i]) i=1 [0,0] representation clause affects compilation only [0,0] T(as_exp_void) + Store(function_call) + Stop T(as_dscrt_range) P(select) as_select_clauses_s + ∑ T(as_set_clauses_s.as_list[i].as_exp_void) comp guards i=1 + {map (λx.Node1 + T(x.as_stm_s) + Node2, as_select_clauses_s.as_list)</pre>
pragma_id pragma_id pragma_s private private_type_id proc_id procedure procedure_call qualified raise range record record_rep rename return reverse select	2 0 n 0 0 1 2 1 2 1 1 2 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2	<pre>[0,0] compiler directive may change global params, but no code gen Timing Assertion pragmas are handled, of course. [0,0] identifier "symbol table" entry [0,0] compiler directive may change global params, but no code gen [0,0] [0,0] identifier "symbol table" entry [0,0] identifier "symbol table" entry [1,0] symbol table" entry [1,0] symbol table " entry [1,0] may be by compiler [1,0] representation clause affects compilation only [1,0] representation clause affects compilation only [2,0] [1,3] spect spect</pre>

select_clause_s	n	subsumed in schema for select node
selected	2	if as_name.sm_obj_type is a record then
		if it has a variant then
		T(as_name) + P(variant_tag_check) + T(as_designator_char)
		else
		(as_name) + 1(as_designator_cnar);
		T(x) = T(x) = T(x)
simple rep	2	$I(as_name) + I(as_uesignawi_cnar),$
shipe_rep	2	[U,0] representation clause anects compliation only
BILCE	2	$1(as_name) + 1(as_name),$ gize = lag deart range $1 \times gm$ even type as constrained of implicities
etm e	n	n
Sum_S	-	TY and listfil)
string literal	0	[0.0]
stub	0	[0,0] Used for separate compilation nurposes only
subprogram body	3	[0,0] = 0000 for separate completion purposes only
subprogram_cocj	3	if as subprogram def + void
subprogram_deci	J	then save id(as designator) = Was subprogram def) + Was header)
euhtune	2	Tree constrained)
subtype id		[0 0] •• identifier "symbol table" entry
subtype_tu	2	[0,0] - Identifier Symbol table entry
teek body	2	P(tesk body eleb) + (reverid(es id & "body") - T(reverid(es id b))
task_body id	2	$\Gamma(\text{task}_\text{body}_\text{elab}) + [\text{save Id}(\text{as}_\text{Id} \text{ c}^{-}\text{body}) - \Gamma(\text{as}_\text{block}_\text{stub})]$
task_bouy_lu	0	[0, v] = Identifier Symbol (able entry D(task man alah) + (rawaid(as id) - T(as task daft))
task_deci	4	$\Gamma(\text{task}_\text{spec}_\text{elab}) + (\text{save Iu}(\text{as}_\text{iu}) = I(\text{as}_\text{task}_\text{uel})) + Oneve Activate(as, id)$
tools anos	1	(Ven deel a)
rask_sbec	1	Activated by allocators or declarations (if declaration, guove estivate it)
torminato	0	P(terminate)
timed entry	2	D(timed ontro)
umed_entry	4	$\Gamma(\text{unded}_\text{entry}) + $ Node1 + $T(\text{agentry} = 2)$ + Node2: first straig a delay stra
		Node1 + $T(as_sum_sz)$ + Node2; first sum is a delay sum Node1 + $T(as_sum_s1)$ + Node2; first straig entry cell
tume	3	Type type energy + Type dearmant yer a)
type	0	- if task type need to save it as well
type id	0	[0 0] identifier "symbol table" entry
universal fixed	0	
universal integer	0	[0,0]
universal real	0	[0,0],
universai_rear	 	[0,0] controls visibility of ids: no code generated
used bltn id	<u> </u>	[0,0] - conditions visionity of idis, no code generated
used bits on	<u>^</u>	
used char	<u>^</u>	
used name id	<u> </u>	[0,0], [0 0]
useu_name_lu	0	
useu_object_iu	<u> </u>	
used_oh	<u>v</u>	[U,U] id al * (Maa time anaa) Maa abiaat dath :
var	3	$10_81 + [1(as_type_spec) + 1(as_object_del) + if as_object_del) + if as_object_del = word then init(as_neme) also store(as_neme) +$
		if as type snot - task snot then (here Activate(as id))
		- similarly for components of as type spec
var id	0	[0 0] identifier "symbol table" entry
varient	2	Type choice a) + Type record)
variant part	2	$T(as_{choice}) + T(as_{choice})$
variant e	<u>~</u>	n
Vallaut_3	"	V TYan lintfil)
		2 1(ao_10(1))
void	0	[0 0] void attribute: no code or sementic value
while	1	Type own)
writh	r	[0 0] - controls visibility of ide: no code concerted

Figure 4: Timing Schema for DIANA Nodes by Node Type
Notes:

- Schema with nodes are also described graphically in the next section.
- Expressions may be static. In this case they are evaluated during compilation and a value for the attribute sm_value is added in the DIANA representation. In this case the expression is handled as a constant object of the appropriate type. In other words, all nodes which can be expressions are first checked for the existence of an sm_value attribute before proceeding with application of the normal schema. These nodes include conversion, qualified, parenthesized, aggregate, binary, membership, indexed, slice, selected, all, attribute, attribute_call, and function_call.
- In loops the phrase n * T(x) is equivalent to unrolling the loop. This is only significant in cases where T(x) introduces nodes. If T(x) is simply additional edge weight, then n*T(x) can be directly computed using multiplication.
- Similarly, branching statements like if and case must be graphically represented if they contain nodes. This is illustrated in the next section.
- The abort statement non-cooperatively cuts off a task from further rendezvous and "marks" it for termination. This is normally used to recover from an error state. In any case, analysis stops on encountering an abort statement (like it does on raise statements) and chooses another parallel path as the worst or best case.
- Some schema of the form [0,0] are actually unreachable in computing the schema formula as defined. In general these are "ID nodes" which represent an identifier or operator of some sort. These nodes are very important in the computation of the auxiliary functions like store, access and init. ID nodes contain the semantic type information these auxiliary functions need. ID nodes are always leaves (i.e., they are never internal nodes) and are meaningful only within the context they appear. Therefore, the schema of their parent node normally include any primitives that context may induce as well as generating any auxiliary function computations necessary.

Event Structure for DIANA Objects

The potential context switches, or events, are introduced by certain Ada program constructs. The corresponding DIANA node types are listed below with a graphic description of the transformation involved. Figure 7 in the next chapter illustrates the transformation with some examples. Or represents a vertex where a context switch may occur. Other vertices are used to connect edges and to gather alternative choices. Loops containing the following structures are unrolled completely if bounded and as far as necessary if unbounded.





branch 1 through n-1 have some sort of node structure, branch n is computed from all of the *sequential* branches as is normally done. If no branch is purely sequential, all n branches have nodes.

In combining the timing graphs into a system network, entry_calls and accepts are *stitched* together like shown in the diagram below. Constructs which have alternative edges are instantiated with exactly one of those edges. Where the choice matters, careful selection must be made based on the analysis being performed. For instance, the worst case for a cond_entry in an unbounded loop is to always choose the *else* part. This results in an infinite chain of *else*'s. (This also illustrates why it is generally bad programming practice to use a conditional entry in an unbounded loop). Most of the time, however, the choices are evident.



Figure 5: Stitching Together an Accept Statement and Entry Call

Assertions

An assertion is simply a statement. Classically, a software developer uses assertions to make claims about the state or nature of the program at a particular point in the code. Often these assertions are embedded as comments in the code itself. A small number of programming languages like Eiffel include certain assertions as part of their syntax [21]. In some cases, an automated tool may process these assertions to generate more powerful claims. This basic idea is applicable to timing analysis. In fact, it grants much of the power for analyzing designs or incomplete code segments. Assertions may be used to bound unbounded loops or recursion, to specify the length of time some code will take without specifying the code itself, and to mark relevant points in the analysis.

A few existing timing analysis techniques use assertions of some sort. Flex [22], like Eiffel, is a program with built-in assertions. As a real-time language, these assertions allow run-time checking of timing behavior. The Flex approach is part of the language definition, however, and this does not help with the analysis of Ada.

Ada provides a handy construct for implementing assertions, as well. They are not checked at run-time, however. The pragma statement does not generate code per se. Instead it passes a directive or a suggestion to the compiler on how to compile the code around it. Except for some standard pragmas, pragmas are considered to be implementation defined. Since they may only change the way code is compiled and not its correctness, a compiler must ignore any pragma it does not recognize (although it may print a warning) [23].

The seven assertions defined for this timing analysis are therefore implemented as pragmas. Each is prefixed with "TA_" to help ensure no conflict with any compiler's own pragma set. Pragma statements are very similar to procedure calls. Like procedure calls, arguments to the pragma may be positionally associated or name associated. Either style may be used except with the TA_Time_By_Primitives assertion which must be name associated.

TA_Loop_Bound (Low, High : natural)	This assertion may appear as the first statement in a loop body. If it does, it defines the range of times that the loop will execute. It overrides the bounds derived by analysis of a for-loop specification.
TA_Recursion_Bound (Low, High : natural)	Similar to a loop bound. It may occur as the statement immediately following a self-recursive procedure or function call. Mutual recursion and recursive call chains are not supported.
TA_Measure_Start	Ignore previous code in this compilation unit for timing analysis purposes. Use this point as an analysis start point. This assertion is intended for use in main programs.
TA_Measure_Stop	Ignore following code in this compilation unit for timing analysis purposes. This assertion is intended to mark where the end event should be inserted.
TA_Time_Absolute (Low, High : Natural)	This assertion is treated exactly like a code sequence which reduces to an edge with time bounds [low, high] (in cycles).
TA_Time_Mix (Instruction_Number : Natural; Mix : Mix_Type)	This assertion is similar to TA_Time_Absolute but uses an instruction count and average instruction time range (mix) to compute the time bounds.
TA_Time_By_Primitives (<prim_name> => Natural,)</prim_name>	Like TA_Time_Mix, this assertion takes instruction counts as its arguments. Instead of counting "average" instructions; however, the developer can specify the number of primitive times. Since named association is used, only the primitives of interest need be listed.

į

Compiler Analysis

In order to determine the values of primitive times, careful analyses of the compiler and the target hardware architecture are required. The compiler analysis must determine the code generated corresponding to each primitive. The hardware architecture analysis must calculate the execution time of this code. Vendor input greatly simplifies the process. However, direct observation of the compiler and hardware may be needed.

The implementation done in conjunction with this thesis uses the Meridian⁵ Macintosh Ada compiler operating on a Macintosh IIsi. Neither the compiler nor the hardware were developed with real-time criteria in mind. This means that predictability is not directly supported and that worst-case times may be significantly worse than average case times.

Hardware analysis must consider the instruction timing of the processor along with system interrupt handlers and bus/processor contention for other system maintenance activities. Vendor timing data is crucial for instruction timing. In the case of the Macintosh, its Motorola 68030 processor is described in [24]. Without timing data, extensive testing with logic analyzers would be necessary to measure either instruction timing or primitive routine timing. These tests could not guarantee bounds on these times unless they can guarantee testing all possible conditions for execution. Adequate vendor data utilizes design knowledge to ensure that time bounds given are true bounds or at least bounds under specified conditions. The Motorola data specifies the worst case execution time under assumptions on the length of

⁵ Meridian Ada[™] 4.1, Meridian Software Systems Inc., 10 Paseur St., Irvine, CA, 92718.

bus cycles and averaging instruction alignment cases. It does not, however, specify best case execution time.

Code generation analysis is also simplified with access to vendor design data. Particularly, a compiler which uses DIANA as an intermediate representation is relatively easy to trace through the code generation phase relative to the schema. The Meridian compiler, however, does not use DIANA and does not supply insight on code generation. Under these circumstances, analysis may be accomplished by disassembly of the compiler libraries and test programs. These test program listings are compared to the source listing and corresponding DIANA structure to associate primitives with measurable code segments. While this approach lacks the same fundamental guarantees as hardware testing, compiler activity is very likely to follow the constraints of the language definition and common compilation practice. Some of these constraints are embedded in the DIANA construction. This helps make the relevant cases more obvious. In the event of a prediction anomaly, however, a new set of code generation circumstances is one of the first things to look for. The primitive times used in this implementation are developed using the disassembly method described here.

Primitive times disassociate the uniqueness of each compiler/hardware /system grouping, but require analysis of each grouping to determine the values of these primitives. Vendor data greatly simplifies the analysis. A production system would need high quality, high reliability predictions. Vendor data is a fundamental necessity to achieve that level. In the experimental implementation developed with this thesis, vendor data is available on the instruction set timing, but experimentation is used to determine system interference and code generation patterns.

Execution Time Prediction Algorithm

Generate the timing graph as described earlier. Choose start events (nodes) of interest. Select the end event of interest. Sum all edges which "precede" the end event in the graph. An edge "follows" an event when no path can be found beginning with a start event and not including the event. An edge that does not follow an event, precedes it.

Note that without priorities, unbounded loops containing conditional entries and select statements with else clauses create busy tasks. Theoretically, these tasks may run indefinitely without relinquishing the processor. Graphically, unrolling these loops create an indefinite number of edges that have no dependencies on other paths (like a rendezvous does). Thus, if the first of these edges precedes the end event, then the entire unrolled loop can precede the end loop. Thus, these constructs must be used carefully, or else, a method other than the simple graph analysis above must be used (such as the CRSM approach defined in the next chapter).

Discussion

If done by hand, applying this graph construction technique is tedious. It needs to be automated. Non-trivial problems generate extremely large DIANA trees. Automating this turned out to be a difficult problem, however. The difficulties were not in the technique; but instead, in the development environment. The task was larger than the system could handle.

On the other hand, this drove home the need for an automated timing analysis technique. Being forced to use this, relatively abstract, method by hand made me realize the extreme difficulty in evaluating the timing characteristics of the program. It also seemed to show why timing analysis is not done as often as it should.

The other issue with developing this approach is the need for compilers and systems which make some effort to be predictable. The Meridian system used unbounded recursion or iteration in several areas. Unless it was clear that some natural bound applied to the value, this created great difficulty. Alarms, in particular, used several cases of recursion and endless loops mostly in searching and deleting. For this reason, they could not be adequately characterized so I deferred investigating things like timed_entries.

The combination of assertions, source code analysis and the concurrency analysis upcoming, provides a broad toolkit to the programmer/analyst who needs to track, verify, or bound the performance of his or her system.

Chapter IV

Concurrency Model

Introduction

The amount of concurrency (both real and perceived) completely changes the timing behavior from one concurrency model to the next. Two common models are the *fully concurrent* model and the *interleaved* model. The applicable model is closely tied to system scheduling decisions as discussed earlier. Ada does not specifically require some degree of concurrency or another. Its model is compatible with either a fully concurrent architecture or an interleaved system, as well as combinations of the two.

This thesis applies to the simple case of interleaved concurrency on a single processor. Currently, most Ada compilers are limited to direct exploitation of a single processor using the Ada language constructs. Multiple processors are sometimes made available through usage of underlying operating system capabilities. Besides wishing to avoid incorporating arbitrary operating system characteristics, multiple processors introduce interprocessor communication contention which greatly complicates timing analysis. Some on-going research is directed at the topic of

predictable interprocessor scheduling and communication [9, 25-27]. Certainly, the trend is toward multiprocessor systems and direct Ada support of these systems. Extending the approach here to support multiprocessors and resource contention is the logical next step in research.

This chapter discusses Ada's rules on concurrency as well as techniques for modelling Ada concurrency. The two techniques presented are a simple PERT technique and Shaw's Communicating Real-Time State Machines (CRSM) [8]. The PERT technique is the one developed in the current schema definitions. It is usable in single-processor systems with no task priorities. CRSM accounts for prioritization and may be extendable to multiprocessing systems. It is more complex to generate and requires automated support to execute, however. This chapter also outlines an approach for incorporating CRSM into the schema developed here.

Ada Concurrency Model

The Ada Concurrency Model is straightforward. It follows closely Hoare's Communicating Sequential Processes [28, 29]. Tasks embody control flows that may execute in parallel. The Ada main program may be considered as a task for this purpose. At no time may a task with a lower priority run if a task with a higher priority is runable on a given processor. Scheduling decisions between tasks of the same priority is left implementation dependent.

This model implies that a context switch between tasks will only occur when a task blocks or a higher or equal priority task becomes runable. A task may block at any of its *synchronization points*. These include the end of its activation, the activation point of another task, an entry call, the start or

the end of an accept statement, a select statement, a delay statement, an exception handler, or an abort statement. It may also block if it uses a blocking system call.⁶ In the worst case, a task blocks in all of these situations and a context switch occurs. A task may become runable as a result of an interrupt or expiration of a delay statement. Again, the worst case is that each of these results in a context switch.

For the purposes of this thesis, the model is simplified. Exception handlers and abort statements are ignored as error control statements. While performance under error conditions may also be critical to real-time behavior, the additional complexity which exception handling and aborts introduce requires further work. Furthermore, blocking system calls are not supported since their behavior is implementation dependent. A more general implementation which accounts for resource contention could add this capability. Finally, interrupts are modelled as one class of starting events; that is, as user specified nodes in the timing graph. The time (or relative time) of a series of interrupts must be supplied and the proper entries graph chosen by the user. Interrupts may also be handled solely by the operating system (i.e., clock tick interrupts). In this case the interrupts are not handled by the Ada program. They are ignored in the timing analysis except as they contribute to system interference.

PERT Networks

PERT networks are analyzed by computing the critical path to reach an end event. Parallel tasks are allowed to execute in parallel. That is, PERT

⁶ In some implementations a blocking system call will block the entire Ada program. This is quite common behavior in Unix where Ada tasks execute as light-weight processes within a Unix process. A system call blocks the entire Unix process.

models full parallelism. Resource levelling is added to PERT analysis to force it not to schedule more parallel activities than resources allow. By specifying that all events use the same resource (CPU) and that there is only one, the PERT analysis begins to model an interleaved system.

The graph supplied for PERT analysis is simply the timing graph with the following modifications. A dummy edge (of zero duration) is added between all dangling activities and the end event. This forces the worst case situation that all events which may possibly precede the end event will do so. Start events must be supplied a time of occurrence. The worst case execution time is then the computed completion time for the end event in whatever time reference was used for specifying the start events. The number of context switching nodes must be counted and multiplied by the worst-case context switch time. This quantity is added to the computed end event time.

This model is limited, however. For instance, PERT cannot handle ORbranching.. OR-branching is when only one graph edge leaving a node is executed in a given instance. This occurs in select and expanded "if or" case statements. With select statements, the choice is driven by the existence of an entry call (or the lack of any). This situation is evident within the structure of the graph. An analyst (or perhaps an automated means) can instantiate the select statements necessary.

There are several nuances with writing rendezvous code. Code can introduce "race" conditions when two tasks call a third which selects between the two call just once. Other difficulties are introduced by rendezvous in a "dependent" context. An independent context is where a select statement can be called an indefinite number of times; or more generally, when the code executed by an entry call is independent of any entry calls by other tasks. If the dependencies in the rendezvous sequence are deterministic, then an analyst may construct the graph in that sequence. When a choice must be made in selecting the caller of particular entry, it may not be evident which choice results in the best or worst time bound. Some choices may not terminate. In the dependent context case, a choice may affect the availability of calls for other choices. In the worst case, all combinations of rendezvous sequences would be tried. Trying all such combinations requires a number of network analyses exponential to the minimum of the number of entry calls and the number of accept points. In almost all cases, programs which may not terminate or which deadlock based on race conditions or which deadlock based on the order of task execution are erroneous.

A simple tasking program illustrates how the PERT technique is used in analyzing timing behavior. The program results in four graphs which must be constructed into a single PERT network. The program is followed by the individual graphs and the resulting network. procedure DOIT is task body TASK B is

```
begin
COUNT A, COUNT B : INTEGER := 0;
                                                  for I in 1 .. 100 loop
RESULT C
                 : INTEGER := 6;
                                                      COUNT B := COUNT B + I;
                                                  end loop;
task TASK A is
                                                  TASK C.ENTRY B;
                                                  COUNT_B := COUNT_B + COUNT_B;
end TASK A;
                                              end TASK B;
task TASK B is
end TASK B;
                                              task body TASK_C is
                                              begin
task TASK C is
                                                  for I in 1 .. 2 loop
   entry ENTRY A:
                                                      select
   entry ENTRY B;
                                                        accept ENTRY A do
   entry DONE;
                                                           RESULT C := RESULT C / 2;
end TASK C;
                                                        end ENTRY A;
                                                     or
task body TASK A is
                                                        accept ENTRY B do
                                                           RESULT C := RESULT_C + 4;
begin
                                                        end ENTRY B;
   for I in 1 .. 1000 loop
       COUNT A := COUNT A + 1;
                                                     end select;
   end loop;
                                                  end loop;
   TASK C.ENTRY A;
                                                  accept DONE;
   COUNT_A := COUNT_A + COUNT_A;
                                              end TASK C;
end TASK A;
                                          begin -- DOIT
                                              TASK C. DONE;
                                          end DOIT;
```

Figure 6: Simple Tasking Program



Figure 7: Timing Graphs for Simple Tasking Program



Figure 8: PERT Network for Simple Tasking Program

In figure 8, the PERT network is simply constructed by matching together the timing graphs for the various tasks and subprograms. The boldface edges are the ones added to construct the network. In this case, they are added at

entry calls and task activations. The triangular node represents the end event. Zero-weighted edges tie the completion of all relevant threads to this event. Note that the entry calls in Task A and B are independent. The network could be constructed with the accepts in Task C reversed. The same edges and nodes would still precede the end event; the order does not matter.

Communicating Real-Time State Machines

Communicating Real-Time State Machines (CRSM) are an executable specification technique [8]. Their key feature is the capability for describing timing properties. They are described by a system model and operational semantics. Each concurrent task is represented by a state machine with synchronous intertask communication. The following paragraphs briefly describe what is developed in the above citation.

The system model is a set of state machines and communication *channels*. The state machines are described by a set of states and transitions. The transitions have labels of the form *guard* \rightarrow *command*. Guards are conditions that must be satisfied before execution of the associated command. Omitting the guard is equivalent to a guard of *true*. The commands may change local variables and/or communicate with other machines. Channels abstractly represent communication between two machines. They are identified by an event name. The event may have parameters associated with it; these are set during the communication.

The operational semantics describe how to transform an input of CRSM's and their channels into a time-sequenced event trace. The basic approach is to construct next event lists for each machine. This is followed by selecting

the earliest event(s) from the various lists. These are executed, time updated, and the process repeated.

Time is represented as a range [min, max] associated with each transition. This is the time it takes for the transition to execute. Communication occurs instantaneously. Also each machine has an associated real-time machine which can be used to model delays as well as get time stamps.

CRSM and Ada Tasking Structures

The basic mapping between Ada and CRSM is to model each Ada task (and master subprogram) as a machine. Transitions represent execution of some statement or sequence of statements. Communication between tasks (entries) are modelled using communication channels. The main change is that channels can have *out* variables passed as actual parameters. The event synchronization does not map exactly to an Ada rendezvous which synchronizes the sender and receiver for some bounded but significant amount of time. Thus, events are used to synchronize both the start and the end of the rendezvous. Furthermore, the calling machine is not allowed any other transitions between the events starting and ending the rendezvous.

Operationally, the timing analysis technique models programs on a single processor. Therefore, CRSMs are executed as described, but instead of running all machines in parallel, only one machine is selected to run at at time. A trace begins with the event of interest and an arbitrary runable machine is selected to run at each "blocking" point. Because of Ada interleave semantics, the order transitions are executed is not important (for independent calling contexts). Priorities can be introduced by making scheduling decisions based on the Ada priority scheme.

Because CRSM and Ada both have roots in Hoare's CSP, it is not surprising that the mapping between them is straightforward. By reintroducing parallelism to the operational semantics, CRSM can simulate multi-processing of Ada programs. This and the ability to make scheduling decisions give CRSM a great deal more power than a simple PERT representation of the program.

Integrating Schema Analysis and CRSM Construction

Constructing timing models using CRSM is not much different than using PERT. Both techniques use graphs. The difference is in the forms of the graph; and, of course, what they mean and how they are analyzed. Thus, the same schema apply except for some of those which construct graph elements. These schema are replaced by the constructs in the following table. Input events are marked with question marks. These correspond to output events with the same name and marked with exclamation points. The input RT?[x] is an input from the real-time machine associated with the task to occur at least x seconds in the future.





For the worst case analysis, the max time is used for each transition. For the best case analysis, the min time is used. The completion of the analysis occurs when the activities triggered by the starting events subside. Note that this model does not require assembly of the generated timing graphs. They are input to the operational CRSM model as is.

Dependent tasking contexts introduce similar problems in this model since the calling order may change based on the length of execution at some point. This change in calling order may result in an overall faster or slower time. Thus, always choosing the max or min time may not generate the worst or best case respectively.

Discussion

The concurrent constructs of Ada require special consideration for timing analysis. On a single processor, only one task can use the processor at a time. This interleaved model means that the execution time of a program is the sum of the time spent in each task with one exception. If any tasks use delays, then it may be possible that no task is runable at some point. This idle time must also be added to the program execution time. PERT analysis can help make this determination.

CRSM provides a more powerful model. Although developed as a specification method, it serves well as a descriptive technique. It can model multiple processors as well as more complex scheduling decisions. Its drawback is the requirement for a specific tool to run the model.

Dependent tasking contexts are system designs where the timing properties are dependent on the order that entry calls occur. Neither PERT nor CRSM can simply determine the bounded execution time of such programs. Trying all possible orders of entry calls is the sure way of finding the bounded execution time; however, this is an exponential growth approach. Further understanding and characterizing the conditions which cause dependent tasking contexts is necessary to determine if a better approach can be developed or if the class of programs can be ruled out.

Chapter V

Experiments

Introduction

Experiments consist of benchmark tests. These tests are compiled and run in a *dedicated* system environment. They are also converted to a DIANA representation which is fed to an automating timing graph generator. The resulting timing graphs are combined into a network and analyzed by hand. The results are adjusted for system interference are compared to the experimentally observed times. In a few cases, both the results and experimental times are compared to worst case times computed using hand analysis techniques like Knuth's.

Setup of the Experiments

As stated in the introduction, the experiments consist of Ada programs which undergo timing analysis, which are timed while executed, and whose prediction and execution results are compared. The benchmark programs come from two sources. The first is the Special Interest Group for Ada of the Association for Computing Machinery (ACM SIGAda) Performance Issues Working Group (PIWG). The second are programs specifically constructed to include a wider selection of language features and exercise some of the capabilities of the analyzer.

The PIWG has constructed a series of benchmarks that measure and compare various features of the Ada language. The benchmarks measure both compilation and execution performance. For these experiments, the best choices are execution benchmarks which test basic sequential language constructs and which test simple tasking situations. Many of these benchmarks are relatively simple, so their object code modules may also be hand analyzed for timing behavior.

The other set of programs are somewhat more complex, but still relatively small (less than 100 SLOC). In both cases the programs are enclosed within an iteration loop. The computer clock is read immediately before the iteration loop and immediately after it completes. The iteration loop may be run several times. In the case of the PIWG benchmarks, the number of iterations varies from one run to the next.

Each program is compiled on a Rational R1000.⁷ The Rational creates a DIANA tree for the program. The DIANA tree is copied into a text file and transferred back to the Macintosh. The Rational represents node identifiers with a long hexadecimal value; pointers in semantic attributes are marked with a caret (^). These values are replaced with a simple integer from the set 1 to n where n is the number of nodes in the file. Semantic pointers are replaced with a similar value with a package id extension (e.g., ^standard.9). If it had been available, this file would be loaded into the timing analysis

⁷ Rational and R1000 are registered trademarks of Rational, 1501 Salado Drive, Mountain View, California, 94043.

program which generates timing graphs for its various program units (i.e., subprograms, tasks, package elaborations). Note that compilation units upon which it depends must be loaded before it; particularly the standard package. The timing graphs which result are printed. Without the analysis program, the timing graphs are generated by hand application of the schema to the DIANA representations. Building networks from these graphs and analyzing them generate the bounded execution time predictions of interest.

The expected execution time is then compared to actual executions. The program is compiled and run on the Macintosh system. No other application programs are run including system extensions like screen savers or virus protection software. Furthermore, keys are not pressed after beginning the test execution and the mouse is not moved. Other programs, particularly system extensions, and input device activity all add to system interference. The programs do not require disk or screen I/O within the critical timing section. The benchmark programs complete by printing out the timing measurements it made.

The execution time of the program (T_{ex}) is represented in the equation:

 $T_{measured} = T_{clock} + T_{loop-overhead} + I \times T_{ex}$ where I is the number of iterations in the loop. Solving this for T_{ex} results in

 $T_{ex} = [T_{measured} - (T_{clock} + T_{loop-overhead})] / I \approx T_{measured} / I$ if I is large enough. Before comparison, the predicted times must be adjusted for the estimated system interference experienced by the test.

System interference is the amount of time the system spends handling interrupts rather than running the program of interest. A simple benchmark measures the amount of time in a simple loop. Running this benchmark under the system configuration described above gives the data necessary to derive a range of nominal system interference. The benchmark was

disassembled and hand analyzed. The result was compared to the timed execution result and a range for system interference determined.

A better experiment was attempted. This would measure the execution time with interrupts disabled and compare it to the normally measured time. The relative difference would represent system interference. The experiment failed because the system routine for reading the hardware clock directly did not work and the other clocks were interrupt driven. Note that disabling interrupts for any significant period of time, in general, would break the Ada run-time system and thus is not feasible for application experiments . However, until the problems mentioned above can be fixed, disabling interrupts is not viable for characterizing system interference, either.

The expected execution time of the system interference benchmark was computed as requiring 228,030,018 - 250,035,023 cycles which at 25 MHz equals 9.12 - 10.00 seconds. The measurement was repeated 100 times. Each measurement was either 14 or 15 seconds. The first time was observed 45 times; thus the weighted average is 14.55 seconds. This means the time spent in interrupt handlers is 4.55 - 5.43 seconds distributed across 60 ticks per second. From this data the average time spent in the tick interrupt handler is 5.21 - 6.22 ms. The nominal system interference, the percentage of each tick spent handling interrupts, is then 31.3 - 37.3%.⁸ Thus, 62.7 - 68.7% of the processor is available for the experiment and the predicted times should be divided by these amounts to give the comparable predictions with system interference accounted for.

⁸ With five system extensions installed, the system interference rises to 60-65%.

Experimental Results

The following table summarizes the results collected. All results are tabulated in Appendix D. Without the automation to generate timing graphs, very few predictions were completed. The predicted times are corrected for system interference.

PIWG	256	1024	8192	256	1024	8192
Experiment	Iterations	Iterations	Iterations	Iterations	Iterations	Iterations
-	Measured	Measured	Measured	Predicted	Predicted	Predicted
C000001 T	27 - 27	108 - 108	n/m			
C000002 T	28 - 29	113 - 113	n/m			
H000004 T	13 - 13	52 - 52	n/m			
P000001 C	1 - 2	5 - 6	41 - 48	1.17 - 1.58	4.67 - 6.31	37.4 - 50.5
P000001 T	1 - 2	5 - 6	48 - 52	1.27 - 2.01	5.08 - 8.02	40.6 - 64.2
P000010 T	2 - 3	8 - 8	69 - 70			
T000001 T	13 - 13	51 - 52	n/m			
T000004 T	30 - 30	119 - 121	n/m			

Table 1: Iterated Experiment	Resul	ts
------------------------------	-------	----

Experiment	Measured Time	Predicted Time
A000091	1.10 - 1.20 ms	
A000092	2.78 - 2.84 ms	
SimpleTasks	11.8 - 14.6	

Table 2:	Single	result	experim	ients
----------	--------	--------	---------	-------

Interpretation

For the experiments completed, the predictions bound the measurements (as shown in figures 9 and 10). This is hardly surprising if the primitives and schema are defined and calculated correctly. The question is then how good the bounds are. The limiting condition on the tightness of the bounds is the tightness of the primitives involved in the prediction.





Some interesting limits arise from this view. Foremost is the observation that the difference between the actual program execution time and the upper bound must be less than the factor between the lower and upper bounds. In the P000001 experiments above, the factor between the bounds are 1.35 for the control loop and 1.58 for the test loop.



Figure 10: A Plot of Experiment P000001 Test Loop Results and Expectations. The second observation is that the factor for the code in a loop body is invariant, i.e., it does not change if the number of loop iterations change. Plotting the upper and lower bounds against iterations of the loop on log-log charts would generate parallel lines. This is the case, in fact, for the graphs above when plotted on a log-log rather than semi-log scale.

The final observation applies when branching and variable loop bounds are ignored. The factor between the bounds of the prediction cannot exceed the largest factor between the bounds of any primitive used in the calculation. This follows from the way a weighted average works. An average cannot exceed the largest of its input data. On the architecture studied, many primitives only vary by 30 - 50%, others vary by factors of four or more. Tight primitive bounds result in tight prediction bounds.

In the experiments above, the worst case bound exceeded the worst case measurement by factors of 5.2% and 33.7% for the control loop and test loop respectively. However, testing may not execute the worst case. With the low number of runs conducted so far these numbers, particularly the second, are probably high. The observation in the previous paragraph, however, indicates that the P000001 code is relatively tight compared to the primitives that may be encountered in other code. Examples involving tasking, delays, or floating point arithmetic have much larger factors between their bounds.

Coverage of the execution results offer another perspective on how good the predicted bounds are. Coverage is the range of test results from the lowest time to the highest time compared to the range of the bounds. For instance, the measurement runs for the P000001 control loop cover iterated 8192 times cover the range 41 - 48 seconds. The bounds are 37.4 and 50.5 seconds. So, the results cover seven seconds of the 13.1 seconds between the bounds or 53%.

Without more testing and study of the implications of these metrics, it is imprudent to draw any conclusions. The metrics of difference factors and coverage help visualize how the predictions stack up to the actual executions. Predictions that generate a factor of 1 between worst case execution and the worst case bound and 100% coverage are obviously ideal. They may be unrealistic; but may also, in fact, be achievable by applying the technique to systems with hardware and compilers developed to generate predictable code.

Chapter VI

Discussion and Future Research

The proposed approach to timing analysis is promising. It satisfies the objectives outlined in Chapter I, but can certainly be improved. Specifically, it does not take advantage of all the context information available to it; it does not handle exception processing or I/O; and it is limited to single processor systems. These are all areas that should be pursued further. Additionally, continuing improvement can be made to tighten the bounds by tightening the primitive time bounds and manipulating the schema to fit better.

In extending this approach it is important to keep the objectives in mind. A particularly difficult one is portability, the applicability of the technique to different hardware and compiler systems. By manipulating the schema to fit more closely what a particular compiler does, the performance with that compiler will improve. It may be incompatible, however, with another compiler. Thus, the schema must continue to conform closely with the language definition.

Park and Shaw has already somewhat considered the tradeoff between tightness of the execution time bounds and portability [30]. They look at large and small atomic blocks represented by their primitive times. This is another way of factoring context into the picture. By defining primitives in

larger terms (such as an entire assignment statement), one can better characterize the compilers behavior. The disadvantage is that many more primitives are needed (a single assignment primitive is not sufficient; one will be needed for each major type and for significant optimization patterns). The goal is to determine a balanced set of primitives and the method for identifying the proper context for using them.

The most difficult part of the language and most implementations is the tasking model and constructs. Better characterization and understanding of tasking activity is a high priority. Here more than any place else, support by the compiler vendor is crucial to understanding the system. Tasking implementations are 10,000's of lines of code. The exercise of characterizing the tasking behavior of the compiler may also benefit the compiler vendor who should discover what parts of the implementation are least predictable and most difficult to characterize.

Another way to distinguish characterizations is using *modes* and *variability*. I use these terms to describe the things which vary execution times from one system or instance to the next. Modes are those things that are fixed at some point in instantiation of the system (e.g., instruction word alignment in the object code image). Variability refers to environmental factors which may continue to vary (e.g., input values, competing workloads, and operator controls). By so categorizing these things, further characterization of the system may become available as modes are fixed for a particular instance.

Modes and variability can be modelled to some degree with random processes and variables. A more general understanding of the system is potentially available using probabilistic models like those in [31] and [32]. Characterizing primitive times as random variables may allow further a

priori characterization of a prediction's factors and ranges. An important distinction that may be provided by studying modes and variability is separation of dependent and independent random variables. These models may allow for reasonable specification and verification of a non-perfect system, e.g., the system that requires 98% availability.

The trickiest problem that must be addressed is multi-processing and shared resources. Gerber and others have started looking at timing analysis in light of resource contention [33]. Tying resource sharing into a general abstract timing analysis model would provide a critical tool to the developers of real-time systems.

I intend to continue exploring this subject as my research area at the US Air Force Academy. Based on personal experience and the literature, systems developers need timing analysis techniques that can apply to the systems they're building today and will be building tomorrow. These techniques must span the entire development cycle, be general enough to use on several projects, and provide reliable performance. Appendices

Appendix A

Timing Primitives for Mac IIsi and Meridian Ada

The following lists the timing primitives as identified by analysis of the Ada language, DIANA representation and Meridian Ada code generation. Where a high or low execution timing bound has been determined for the primitive, the value is shown in cycles. The Mac IIsi executes at 25 MHz.

. .

accept			attr_count
access			attr_delta
activ	184	618	attr_digits
activation			attr_emax
allocate_main			attr_epsilon
and_then			attr_first
array_access			attr_first
array_aggr_comp_access			attr_first(N)
array_aggr_setup			attr_first_bit
array_cat			attr_fore
array_comp_access			attr_image
array_comp_store			attr_large
array_ge			attr_last
array_greater			attr_last(N)
array_in			attr_last_bit
array_le			attr_length
array_lesser			attr_length(N)
array_not_in			attr_machine_emax
array_store			attr_machine_emin
attr_address			attr_machine_mantissa
attr_aft			attr_machine_overflows
attr_base			attr_machine_radix
attr_callable			attr_machine_rounds
attr_constrained			attr_mantissa

61			
attr_pos	fixed_greater		
attr_position	fixed_identity		
attr_pred	fixed_in		
attr_range	fixed_le		
attr_range(N)	fixed_lesser		
attr_safe_emax	fixed_minus		
attr_safe_large	fixed_mul		
attr_safe_small	fixed_neq		
attr_size	fixed_negation		
attr_small	fixed_not_in		
attr_storage_size	fixed_plus		
attr_succ	fixed_store		
attr_terminated	float_access		
attr_val	float_abs		
attr_value	float_div		
attr_width	float_eq		
ba_and	float_exp		
ba_not	float_identity		
ba_or	float_in		
ba_xor	float_minus		
bool_access	float_mul		
bool_and	float_neq		
bool_eq	float_negation		
bool_neg	float_not_in		
bool_not	float_plus		
bool_or	float_store		
bool_store	for_end	14	22
bool_xor	for_iter	34	50
cond_entry	for_loop	6	36
case	function_access	0	0
context switch	function_call		
convert_array	function_end		
convert_derived	if		
convert_fixed2float	int_abs		
convert_fixed2int	int_div		
convert_float2fixed	int_eq		
convert_float2int	int_exp		
convert_int2fixed	int_ge		
convert_int2float	int_greater		
default_param	int_identity		
delay	int_in		
else	int_le		
fixed_abs	int_lesser		
fixed_access	int_literal_access	0	6
fixed_div	int_minus		
fixed_eq	int_mod		
fixed_ge	int_neq		

int_negation		
int_not_in		
int_plus	8	12
int_rem		
int_times		
integer_access	4	16
integer_store	5	18
iter		
loop		
loop_end		
null		
or_else		
package_elaboration		
procedure_call ⁹	40	858
procedure_end	28	100
queue_entry	992	2082
range_check		
real_literal_access		
record		
record_aggr_comp_acces		
record_aggr_setup		
record_comp_access		
record_comp_store		
rendevous		
select		
slice_access		
slice_store		
string_literal_access		
task_body_elab	92	158
task_spec_elab	2202	4166
terminate	24	94
timed_entry		
variant_tag_check		ļ

⁹ Worst case ame is only 301 when the procedure contains no tasks.

Appendix B

Test Program Source Code

A000001 with TEXT_IO ; use TEXT_IO ;

package DURATION_IO is new FIXED_IO (DURATION) ;

A000018

-- This is a universal Ada function to get CPU time in seconds -- of type DURATION on non time_sharring systems where a -- tailored CPU_TIME_CLOCK is not reasonable -- Do not cross a midnight boundry -- It is modified to read the clock using the Mac OS clock routine rather -- than the calendar package. This gives 1/60th second rather than 1 sec -- resolution. with EVENTS; with MAC_TYPES; function CPU_TIME_CLOCK return DURATION is MaxTicks : Constant := 60 * 86400; -- Duration'last NOW : MAC_TYPES.LONGINT := EVENTS.TICKCOUNT ; begin return DURATION (FLOAT (NOW mod maxTicks) / 60.0); end CPU_TIME_CLOCK ; A000021 package REMOTE_GLOBAL is -- for explicit control of optimization a constant 1 that can not be optimized away A_ONE : INTEGER; --A_ONE is intentionally visible. DO NOT CHANGE IT - -- -GLOBAL : INTEGER := 1 ; -- global object can not be optimized away - -GLOBAL is changed by measurement programs the initialization to 1 is used in the body - -- but could be changed by elaboration order

procedure REMOTE; -- do to calls to this procedure, no compiler -- can optimize away the computation an GLOBAL -- procedure CHECK_TIME (TEST_DURATION : in DURATION) ; -- Just print message if TEST_DURATION less then -- 100 * SYSTEM.TICK or DURATION'SMALL -- end REMOTE_GLOBAL;

```
63
```
A000022 with SYSTEM, TEXT_IO;

```
package body REMOTE_GLOBAL is -- must be compiled last
                                 for explicit control of optimization
  LOCAL : INTEGER;
  procedure REMOTE is -- this is an optimization control procedure
  beain
   GLOBAL := GLOBAL + LOCAL; -- be sure procedure is not optimized away
  exception
   when NUMERIC_ERROR =>
     REMOTE ; -- can not happen if test is working ( prevents inlining )
  end REMOTE;
  procedure CHECK_TIME ( TEST_DURATION : in DURATION ) is
  beain
   if TEST_DURATION < 100 * DURATION'SMALL or
       TEST_DURATION < 100 * SYSTEM.TICK then
      TEXT_IO.PUT_LINE ( " ***** TEST_DURATION not large compared to "
                         & "DURATION'SMALL or SYSTEM.TICK " );
   end if :
  end CHECK_TIME ;
begin
 A_ONE := 1; -- must not be changed by measurement programs
 LOCAL := GLOBAL - A_ONE; -- really a zero but compiler doesn't know
end REMOTE_GLOBAL;
-- This is the ITERATION_COUNT control package for feature measurements
-- The set of procedures provide the automatic stabilizing of the
-- timing measurement. The measurement CPU time must be greater than:
-- 1.0 second, DURATION'SMALL * 100 , SYSTEM.TICK * 100
--
```

-- Note: If there is no control loop, the START_CONTROL and STOP_CONTROL -- do not need to be called.

package ITERATION is -- A000031.ADA

```
subtype ITERATION_COUNTS is INTEGER range 1 .. 32768;
procedure START_CONTROL ;
procedure STOP_CONTROL ( GLOBAL : INTEGER ;
CHECK : INTEGER );
procedure START_TEST ;
procedure STOP_TEST ( GLOBAL : INTEGER ;
CHECK : INTEGER );
procedure FEATURE_TIMES ( CPU_TIME : out DURATION ;
WALL_TIME : out DURATION );
procedure INITIALIZE ( ITERATION_COUNT : out INTEGER );
procedure TEST_STABLE ( ITERATION_COUNT : in out INTEGER ;
STABLE : out BOOLEAN );
end ITERATION ;
```

```
A000033
-- Iteration control package body ( for test development )
- -
     This version is instrumented and may interefere with some
     types of tests
with CPU_TIME_CLOCK ; -- various choices on tape
with CALENDAR ; -- used for WALL clock times
with SYSTEM ; -- used to get value of TICK with TEXT_IO ; -- only for diagnostics
with DURATION IO :
package body ITERATION is -- A000032.ADA
-- CPU time variables
  CONTROL_TIME_INITIAL : DURATION ; -- sampled from CPU_TIME_CLOCK at beginning
  CONTROL_TIME_FINAL : DURATION ; -- sampled from CPU_TIME_CLOCK at end
CONTROL_DURATION : DURATION ; -- (FINAL-INITIAL) the measured time in seconds
  TEST_TIME_INITIAL : DURATION ; -- ditto for TEST
  TEST_TIME_FINAL : DURATION ;
  TEST_DURATION : DURATION ;
-- WALL time variables
  WALL_CONTROL_TIME_INITIAL : DURATION ; -- sampled from CLOCK at beginning
  WALL_CONTROL_TIME_FINAL : DURATION ; -- sampled from CLOCK at end
  WALL_CONTROL_DURATION : DURATION ; -- (FINAL-INITIAL) measured time in seconds WALL_TEST_TIME_INITIAL : DURATION ; -- ditto for TEST
  WALL_TEST_TIME_FINAL : DURATION ;
  WALL_TEST_DURATION : DURATION ;
  MINIMUM_TIME : DURATION := 1.0 ; -- required minimum value of test time
  TEMP_TIME : FLOAT ; -- for scaling to microseconds
  ITERATION_COUNT : ITERATION_COUNTS ; -- change to make timing stable
  CHECK : INTEGER ; -- saved from STOP_TEST call for scaling
  procedure START_CONTROL is
  begin
    CONTROL_TIME_INITIAL := CPU_TIME_CLOCK ;
     WALL_CONTROL_TIME_INITIAL := CALENDAR.SECONDS(CALENDAR.CLOCK) ;
  end START_CONTROL ;
  procedure STOP_CONTROL ( GLOBAL : INTEGER ;
                             CHECK : INTEGER ) is
  begin
     CONTROL_TIME_FINAL := CPU_TIME_CLOCK ;
     CONTROL_DURATION := CONTROL_TIME_FINAL - CONTROL_TIME_INITIAL ;
    WALL_CONTROL_TIME_FINAL := CALENDAR.SECONDS(CALENDAR.CLOCK) ;
    WALL_CONTROL_DURATION := WALL_CONTROL_TIME_FINAL
                               WALL_CONTROL_TIME_INITIAL ;
    if CHECK /= GLOBAL then
       TEXT_IO.PUT_LINE ( " Fix control loop before making measurements." );
       TEXT_IO.PUT_LINE ( INTEGER'IMAGE ( GLOBAL ) & " = GLOBAL " ) ;
       raise PROGRAM_ERROR ;
    end if ;
    TEXT_IO.PUT_LINE ( "Iteration " & INTEGER'IMAGE ( ITERATION_COUNT ) );
    DURATION_IO.PUT ( CONTROL_TIME_INITIAL );
    DURATION_IO.PUT ( CONTROL_TIME_FINAL );
    DURATION_IO.PUT ( CONTROL_DURATION );
    TEXT_IO.NEW_LINE ;
  end STOP_CONTROL ;
  procedure START_TEST is
  beain
    TEST_TIME_INITIAL := CPU_TIME_CLOCK ;
    WALL_TEST_TIME_INITIAL := CALENDAR.SECONDS(CALENDAR.CLOCK) ;
  end START_TEST ;
  procedure STOP_TEST ( GLOBAL : INTEGER ;
                          CHECK : INTEGER ) is
  begin
```

```
TEST_TIME_FINAL := CPU_TIME_CLOCK ;
    TEST_DURATION := TEST_TIME_FINAL - TEST_TIME_INITIAL ;
    WALL_TEST_TIME_FINAL := CALENDAR.SECONDS(CALENDAR.CLOCK)
    WALL_TEST_DURATION := WALL_TEST_TIME_FINAL - WALL_TEST_TIME_INITIAL ;
- -
    ITERATION.CHECK := CHECK ;
    if CHECK /= GLOBAL then
      TEXT_IO.PUT_LINE ( " Fix test loop before making measurements." ) ;
      TEXT_IO.PUT_LINE ( INTEGER'IMAGE ( GLOBAL ) & " = GLOBAL " ) ;
      raise PROGRAM_ERROR :
    end if ;
  end STOP_TEST ;
  procedure FEATURE_TIMES ( CPU_TIME : out DURATION ;
                            WALL_TIME : out DURATION ) is
  beain
--
    compute scaled results
--
    begin
      TEMP_TIME := FLOAT ( TEST_DURATION - CONTROL_DURATION ) ;
      TEMP_TIME := (1_000_000.0 * TEMP_TIME) /
                   ( FLOAT ( ITERATION_COUNT ) * FLOAT (CHECK) );
      CPU_TIME := DURATION ( TEMP_TIME ) ;
    exception
      when others => -- bail out if trouble in conversion
        CPU_TIME := 0.0 ;
    end;
- -
    begin
      TEMP_TIME := FLOAT ( WALL_TEST_DURATION - WALL_CONTROL_DURATION ) ;
      TEMP_TIME := (1_000_000.0 * TEMP_TIME) /
                   ( FLOAT ( ITERATION_COUNT ) * FLOAT (CHECK) );
      WALL_TIME := DURATION ( TEMP_TIME ) ;
    exception
      when others =>
        WALL_TIME := 0.0 ;
    end;
  end FEATURE_TIMES ;
  procedure INITIALIZE ( ITERATION_COUNT : out INTEGER ) is
  beain
    ITERATION_COUNT := 1 ;
    ITERATION.ITERATION_COUNT := 1 ;
  end INITIALIZE :
  procedure TEST_STABLE ( ITERATION_COUNT : in out INTEGER ;
                          STABLE : out BOOLEAN ) is
  begin
    if TEST_DURATION > MINIMUM_TIME then
      STABLE := TRUE ;
    elsif ITERATION_COUNT >= 16384 then
      TEXT_IO.PUT_LINE ( "***** INCOMPLETE MEASUREMENT *****" ) ;
      STABLE := TRUE ;
    else
      ITERATION_COUNT := ITERATION_COUNT + ITERATION_COUNT ;
      ITERATION.ITERATION_COUNT := ITERATION_COUNT ;
      STABLE := FALSE ;
    END IF;
  end TEST_STABLE ;
- -
begin
  if SYSTEM.TICK * 100 > MINIMUM_TIME then
   MINIMUM_TIME := SYSTEM.TICK * 100 ;
  end if:
  if DURATION'SMALL * 100 > MINIMUM_TIME then
    MINIMUM_TIME := DURATION'SMALL * 100 ;
```

end if;

-- MINIMUM_TIME is now the larger of 1.0 second, -- 100*SYSTEM.TICK, -- 100*DURATION'SMALL CONTROL_DURATION := 0.0 ; WALL_CONTROL_DURATION := 0.0 ;

end ITERATION ;

A000091

- -- -"DHRYSTONE" Benchmark Program ----- ----------------Version ADA/1 _ _ - -- ---Date: 04/15/84 --- -Author: Reinhold P. Weicker ---------- As published in Communications of ACM, October 1984 Vol 27 No 10 - ------- -- --- The following program contains statements of a high-level programming -- language (Ada) in a distribution considered representative: --- -- -53% - assignments - control statements 32% ---- procedures, function call 15% --_ _ -- 100 statements are dynamically executed. The program is balanced with - --- respect to the three aspects: - -- -- -- -- statement type - ---- operand type (for simple data types) - -- operand access - -- -- operand global, local, parameter, or constant. ---- --- The combination of these three aspects is balanced only approximately. ----- The program does not compute anything meaningful, but it is syntactically ---- and semantically correct. All variables have a value assigned to them ------ before they are used as a source operand ----package global_def is -- alobal definintions type Enumeration is (ident_1,ident_2,ident_3,ident_4,ident_5); subtype one_to_thirty is integer range 1..30; subtype one_to_fifty is integer range 1..50; subtype capital_letter is character range 'A'..'Z'; type String_30 is array(one_to_thirty) of character; pragma pack(string_30); type array_1_dim_integer is array (one_to_fifty) of integer; type array_2_dim_integer is array (one_to_fifty, one_to_fifty) of integer; type record_type(discr:enumeration:=ident_1); type record_pointer is access record_type; type record_type(discr:enumeration:=ident_1) is record pointer_comp: record_pointer; case discr is when ident_1 =. -- only this variant is used, -- but in some cases discriminant -- checks are necessary enumeration; enum comp: int_comp: one_to_fifty; string_comp: string_30; when ident_2 => enum_comp_2: enumeration; string_30; string_comp_2:

```
when others =>
          char_comp_1,
          char_comp_2:
                                character;
        end case;
    end record;
end global_def;
    with global_def;
    use global_def;
package pack_1 is
  -----
    procedure proc_0;
    procedure proc_1(pointer_par_in: in
                                              record_pointer);
    procedure proc_2(int_par_in_out: in out one_to_fifty);
    procedure proc_3(pointer_par_out: out
                                              record_pointer);
    int_glob: integer;
end pack_1;
    with global_def;
    use global_def;
package pack_2 is
       -----
    procedure proc_6 (enum_par_in:
                                        in
                                                enumeration;
                      enum_par_out:
                                        out
                                                enumeration);
    procedure proc_7 (int_par_in_1,
                                                one_to_fifty;
                      int_par_in_2:
                                        in
                      int_par_out:
                                        out
                                                one_to_fifty);
    procedure proc_8 (array_par_in_out_1: in out array_1_dim_integer;
                      array_par_in_out_2: in out array_2_dim_integer;
                      int_par_in_1,
                      int_par_in_2:
                                          in
                                                 integer);
    function func_1 (char_par_in_1,
                      char_par_in_2:
                                          in
                                                 capital_letter)
                                                   return enumeration;
    function func_2 (string_par_in_1,
                      string_par_in_2:
                                          in
                                                string_30)
                                                   return boolean;
end pack_2;
with global_def, pack_1;
use global_def;
procedure A000091 is -- Dhrystone
 -------------
begin
                        -- proc_0 is actually the main program, but it is
   pack_1.proc_0;
                        -- part of a package, and a program within a
                        -- package can not be designated as the main
                        -- program for execution. Therefore proc_0 is
                        -- activated by a call from "main".
end A000091 ;
with global_def,pack_2;
use global_def;
with cpu_time_clock;
with text_io;
with duration_io;
package body pack_1 is
```

```
69
```

```
bool_glob:
                        boolean;
    char_glob_1,
    char_glob_2:
                        character;
    array_glob_1:
                        array_1_dim_integer;
    array_glob_2:
                        array_2_dim_integer;
    pointer_glob,
    pointer_glob_next: record_pointer;
    start_time : duration ;
    stop_time : duration ;
    iteration_count : constant := 10_000 ;
    procedure proc_4;
    procedure proc_5;
procedure proc_0
is
    int_loc_1,
    int_loc_2,
    int_loc_3:
                        one_to_fifty;
    char_loc:
                        character;
    enum_loc:
                        enumeration;
    string_loc_1,
    string_loc_2:
                        string_30;
    begin
        -- initializations
        pack_1.pointer_glob_next := new record_type;
        pack_1.pointer_glob := new record_type
                             '(
                             pointer_comp => pack_1.pointer_glob_next,
                             discr
                                         => ident_1,
                             enum_comp
                                         => ident_3,
                             int_comp
                                          => 40,
                             string_comp => "DHRYSTONE PROGRAM, SOME STRING"
                                );
        string_loc_1 := "DHRYSTONE PROGRAM, 1'ST STRING";
-----
-- start timer here
     start_time := cpu_time_clock ;
     for i in 1 .. iteration_count loop
        proc_5;
        proc_4;
         -- char_glob_1 = 'A', char_glob_2 = 'B', bool_glob = false
        int_loc_1 := 2;
        int_loc_2 := 3;
        string_loc_2 := "DHRYSTONE PROGRAM, 2'ND STRING";
        enum_loc := ident_2 ;
        bool_glob := not pack_2.func_2( string_loc_1, string_loc_2);
        -- bool_glob = true
        while int_loc_1 < int_loc_2 loop --loop body executed once</pre>
                    pragma TA_LOOP_BOUNDS(1,1);
           int_loc_3 := 5 * int_loc_1 - int_loc_2;
           -- int_loc_3 = 7
           pack_2.proc_7(int_loc_1, int_loc_2, int_loc_3);
            -- int_loc_3 = 7
           int_loc_1 := int_loc_1 + 1;
        end loop;
           -- int_loc_1 = 3
        pack_2.proc_8(array_glob_1,array_glob_2,int_loc_1,int_loc_3);
          -- int_glob = 5
        proc_1(pointer_glob);
        for char_index in 'A'..Char_glob_2 loop --loop body executed twice
            if enum_loc = pack_2.func_1(char_index,'C')
            then -- not executed
```

```
70
```

```
pack_2.proc_6(ident_1,enum_loc);
            end if;
        end loop;
          -- enum_loc = ident_1
          -- int_loc = 3, int_loc_2 = 3, int_loc_3 = 7
        int_loc_3 := int_loc_2 * int_loc_1;
        int_loc_2 := int_loc_3 / int_loc_1;
        int_loc_2 := 7 * ( int_loc_3 - int_loc_2 ) - int_loc_1;
        proc_2(int_loc_1);
     end loop ;
     stop_time := cpu_time_clock ;
     text_io.new_line;
     text_io.new_line;
     text_io.put_line("Test Name: A000091
                                                        Class Name: Composite");
                        ");
     text_io.put("
     duration_io.put((stop_time-start_time)*1000/iteration_count);
     text_io.put_line(" is time in milliseconds for one Dhrystone");
     text_io.put_line("Test Description:");
     text_io.put_line(" Reinhold P. Weicker's DHRYSTONE composite benchmark");
     text_io.new_line;
-- stop timer here
-----
        . . . . . . . . . . . .
end proc_0;
procedure proc_1(pointer_par_in: in record_pointer) is -- executed once
        next_record: record_type
            renames pointer_par_in.pointer_comp.all; -- pointer_glob_next.all
begin
        next_record :=pointer_glob.all;
        pointer_par_in.int_comp := 5;
        next_record.int_comp := pointer_par_in.int_comp;
        next_record.pointer_comp:= pointer_par_in.pointer_comp;
        proc_3(next_record.pointer_comp);
            -- next_record.pointer_glob.pointer_comp = pointer_comp.next
        if next_record.discr = ident_1
        then -- executed
            next_record.int_comp := 6;
            pack_2.proc_6(pointer_par_in.enum_comp,next_record.enum_comp);
            next_record.pointer_comp := pointer_glob.pointer_comp;
            pack_2.proc_7(next_record.int_comp,10,next_record.int_comp);
        else
            pointer_par_in.all := next_record;
        end if;
end proc_1;
procedure proc_2 ( int_par_in_out: in out one_to_fifty)
is -- executed once
   -- in_par_in_out = 3 becomes 7
    int_loc : one_to_fifty;
    enum_loc : enumeration;
begin
    int_loc := int_par_in_out + 10;
    loop
                 pragma TA_LOOP_BOUNDS(1,2);
        if char_glob_1 = 'A'
        then
            int_loc := int_loc - 1;
            int_par_in_out := int_loc - int_glob;
            enum_loc := ident_1; -- true
        end if;
    exit when enum_loc = ident_1; -- true
    end loop;
end proc_2;
procedure proc_3(pointer_par_out: out record_pointer)
is -- executed once
```

```
-- pointer_par_out becomes pointer_glob begin
```

```
if pointer_glob /= null
```

```
71
```

```
then -- executed
       pointer_par_out := pointer_glob.pointer_comp;
    else
        int_glob := 100;
    end if;
    pack_2.proc_7(10, int_glob, pointer_glob.int_comp);
end proc_3;
procedure proc_4
ίs
    bool_loc : boolean;
begin
    bool_loc := char_glob_1 = 'A';
    bool_loc := bool_loc or bool_glob;
    char_glob_2 := 'B';
end proc_4;
procedure proc_5
is
begin
    char_glob_1 := 'A';
    bool_glob := false;
end proc_5;
end pack_1;
    with global_def,pack_1; use global_def;
package body pack_2 is
function func_3(enum_par_in: in enumeration) return boolean;
        -- forward declaration
procedure proc_6(enum_par_in: in enumeration;
                 enum_par_out: out enumeration) is
begin
    enum_par_out := enum_par_in;
    if not func_3(enum_par_in) then
       enum_par_out := ident_4;
   end if;
    case enum_par_in is
        when ident_1 =>enum_par_out := ident_1;
        when ident_2 =>if pack_1.int_glob>100
                        then enum_par_out := ident_1;
                        else enum_par_out := ident_4;
                        end if;
        when ident_3 =>enum_par_out := ident_2; -- executed
        when ident_4 =>null;
        when ident_5 =>enum_par_out := ident_3;
   end case;
end proc_6;
procedure proc_7(int_par_in_1,
                int_par_in_2: in one_to_fifty;
                int_par_out: out one_to_fifty) is
int_loc : one_to_fifty;
begin
  int_loc := int_par_in_1 + 2;
  int_par_out := int_par_in_2 + int_loc;
end proc_7;
procedure proc_8 (array_par_in_out_1: in out array_1_dim_integer;
                  array_par_in_out_2: in out array_2_dim_integer;
                  int_par_in_1,
                  int_par_in_2:
                                      in integer)
is
int_loc: one_to_fifty;
begin
   int_loc := int_par_in_1 + 5;
    array_par_in_out_1(int_loc) := int_par_in_2;
    array_par_in_out_1(int_loc + 1) :=
                                        array_par_in_out_1(int_loc);
    array_par_in_out_1(int_loc + 30) := int_loc;
```

```
72
```

```
for int_index in int_loc..int_loc + 1 loop -- loop body executed twice
                  pragma TA_LOOP_BOUNDS(2,2);
         array_par_in_out_2(int_loc,int_index) := int_loc ;
    end loop;
    array_par_in_out_2(int_loc,int_loc-1) :=
                            array_par_in_out_2(int_loc, int_loc-1) + 1;
    array_par_in_out_2(int_loc + 20, int_loc) :=
                            array_par_in_out_1(int_loc);
    pack_1.int_glob := 5;
end proc_8;
function func_1 (char_par_in_1,
                 char_par_in_2: in capital_letter) return enumeration
ίs
char_loc_1, char_loc_2 : capital_letter;
begin
    char_loc_1 := char_par_in_1;
    char_loc_2 := char_loc_1;
    if char_:oc_2 /= char_par_in_2 then
        return ident_1;
    else
        return ident_2;
    end if;
end func_1;
function func_2(string_par_in_1,
                 string_par_in_2: in string_30) return boolean
is
int_loc: one_to_thirty;
char_loc: capital_letter;
hegin
    int_loc := 2;
    while int_loc <= 2 loop</pre>
                  pragma TA_LOOP_BOUNDS(1,1);
         if func_1(string_par_in_1(int_loc),
                  string_par_in 2(int_loc+1)) = ident_1 then
             char_loc := 'A';
             int_loc := int_loc + 1;
        end if;
    end loop;
    if char_loc >='W' and char_loc < 'Z' then
        int_loc := 7;
    end if;
    if char_loc = 'X' then
        return true;
    else
        if string_par_in_1 > string_par_in_2 then
            int_loc := int_loc + 7;
            return true;
        else
            return false;
        end if;
    end if;
end func_2;
function func_3(enum_par_in: in enumeration) return boolean
15
    enum_loc: signaration;
begin
    enum_loc = enum_par_in;
    if enum_loc = ident_3 then
       return true;
    end if;
end func_3;
end pack_2;
-- Ada version of Whetstone Beachmark Program
```

```
73
```

- This must be edited to "with" the compiler suppliers math routines
 SIN, COS, ATAN, SQRT, EXP and LOG
 These results may be interesting to compare to Z000093 that uses
 a physically included, all Ada set of math routines

_____ - distributed as A000092.ADA WHETADA.ADA - ------ ------Ada version of the Whetstone Benchmark Program. Reference: "Computer Journal" February 1976, pages 43-49 for description of benchmark and ALGOL60 version. - -- ----- --- Note: Procedure POUT is omitted. - -- --- From Timing Studies using a synthetic Whetstone Benchmark by Sam Harbaugh and John A. Forakis - -- -- -- --- Authors Disclaimer The Whetstone measure deals only with the most basic scientific/ ----- ' -- computational aspects of the languages and computers and no general ---- conclusions should be drawn from this work. Application specific -- benchmarks should be written and run by anyone needing to draw - --- conclusions reguarding suitability of languages, compilers and ----- hardware. This data is reported to stimulate interest and work in ---- run time benchmarking and in no way is meant to influence anyone's ---- choice of languages or software in any situation " - -with CPU_TIME_CLOCK ; with TEXT_IO; use TEXT_IO; -- Change the following line to use the compiler vendors or manufacturers -- math library. with MATH_LIB; use MATH_LIB; -- manufacturers routines (Meridian for Mac) procedure A000092A is --pragma SUPPRESS(ACCESS_CHECK); DO NOT USE PRAGMA SUPPRESS for PIWG --pragma SUPPRESS(DISCRIMINANT_CHECK); --pragma SUPPRESS(INDEX_CHECK); --pragma SUPPRESS(LENGTH_CHECK); --pragma SUPPRESS(RANGE_CHECK); --pragma SUPPRESS(DIVISION_CHECK); --pragma SUPPRESS(OVERFLOW_CHECK); --pragma SUPPRESS(STORAGE_CHECK); --pragma SUPPRESS(ELABORATION_CHECK); package REAL_IO is new FLOAT_IO(FLOAT); use REAL_IO; subtype CYCLES is INTEGER range 10..50; procedure WHETSTONE(NO_OF_CYCLES : in CYCLES; START_TIME, STOP_TIME: out FLOAT) is -- Calling procedure provides -- the encompassing loop count, NO_OF_CYCLES. type VECTOR is array (INTEGER range <>) of FLOAT; X1, X2, X3, X4, X, Y, Z : FLOAT; E1 : VECTOR(1..4); J,K,L : INTEGER; -- Set constants T : constant := 0.499975; T1 : constant := 0.50025; T2 : constant := 2.0;-- Compute the execution frequency for the benchmark modules N1 : constant := 0;--Module 1 not executed N2 : constant := 120; N3 : constant := 140; N4 : constant := 3450; N5 : constant := 0; -- Module 5 not executed N6 : constant := 2100; N7 : constant := 320; N8 : constant := 8990; **N9** : constant := 6160; N10: constant := 0; -- Module 10 not executed N11: constant := 930;

```
procedure PA(E: in out VECTOR) is
-- tests computations with an array as a parameter
   ] : INTEGER;
   -- T,T2 : FLOAT are global variables
   begin
     J:=0;
     loop
                              pragma TA_LOOP_BOUNDS(6,6);
                              E(1) := (E(1) + E(2) + E(3) - E(4)) * T;
                              \begin{array}{l} E(2) := (E(1) + E(2) - E(3) + E(4)) * T; \\ E(3) := (E(1) - E(2) + E(3) + E(4)) * T; \end{array}
                              E(4) := (-E(1) + E(2) + E(3) + E(4)) / T2;
                              J := J + 1;
                              exit when j \ge 6;
                    end loop;
end PA;
procedure P0 is
-- tests computations with no parameters
-- T1,T2 : FLOAT are global
-- E1 : VECTOR(1..4) is global
-- J,K,L : INTEGER are global
   begin
     E_1(J) := E_1(K);
     E1(K) := E1(L);
     E1(L) := E1(J);
   end P0;
procedure P3(X,Y: in out FLOAT; Z : out FLOAT) is
-- tests computations with simple identifiers as parameters
-- T,T2 : FLOAT are global
   begin
     X := T * (X + Y);
Y := T * (X + Y);
     Z := (X + Y) / T2;
   end P3;
begin
  START_TIME := FLOAT(CPU_TIME_CLOCK); --Get Whetstone start time
  CYCLE_LOOP:
  for CYCLE_NO in 1..NO_OF_CYCLES loop
     -- Module 1 : computations with simple identifiers
        X1 := 1.0;
        X2 := -1.0;
        X3 := -1.0;
         X4 := -1.0;
         for I in 1..N1 loop
            X1 := (X1 + X2 + X3 - X4) * T;
            X2 := (X1 + X2 - X3 + X4) * T;
            X3 := (X1 + X2 + X3 + X4) * T;
            X4 := (-X1 + X2 + X3 + X4) * T;
         end loop;
     -- end Module 1
     -- Module 2: computations with array elements
```

 $\begin{array}{l} E1(1) := (E1(1) + E1(2) + E1(3) - E1(4)) * T; \\ E1(2) := (E1(1) + E1(2) - E1(3) + E1(4)) * T; \\ E1(3) := (E1(1) - E1(2) + E1(3) + E1(4)) * T; \\ E1(4) := (-E1(1) + E1(2) + E1(3) + E1(4)) * T; \end{array}$

E1(1) := 1.0; E1(2) := -1.0; E1(3) := -1.0; E1(4) := -1.0; for I in 1..N2 loop

end loop; -- end Module 2

```
-- Module 3 : passing an array as a parmeter
   for I in 1...N3 loop
        PA(E1);
   end loop;
-- end Module 3
-- Module 4 : performing conditional jumps
   J := 1;
   for I in 1...N4 loop
      if J=1 then
          J := 2;
      else
          J := 3;
      end if;
      if J>2 then
          J := 0;
      else
          J := 1;
      end if;
      if J<1 then
          J := 1;
      else
          J := 0;
      end if;
   end loop;
--end Module 4
-- Module 5 : omitted
-- Module 6 : performing integer arithmetic
   J := 1;
   K := 2;
   L := 3;
   for I in 1.. N6 loop
     J := J * (K-J) * (L-K);
K := L*K - (L-J) * K;
      L := (L-K) * (K+J);
      E1(L-1) := FLOAT()+K+L);
      E1(K-1) := FLOAT(J*K*L);
   end loop;
-- end Module 6
-- Module 7 : performing computations using trigonometric
- -
              functions
  X := 0.5;
   Y := 0.5;
   for I in 1..N7 loop
    X := T^*ATAN(T2^*SIN(X)^*COS(X)/(COS(X+Y)+COS(X-Y)-1.0));
     Y := T*ATAN(T2*SIN(Y)*COS(Y)/(COS(X+Y)+COS(X-Y)-1.0));
   end loop;
-- end Module 7
-- Module 8 : procedure calls with simple identifiers as
- -
              parameters
  X := 1.0;
   Y := 1.0;
   Z := 1.0;
   for I in 1...N8 loop
     P3(X,Y,Z);
   end loop;
-- end Module 8
-- Module 9 : array reference and procedure calls with no
- -
              parameters
   ) := 1;
   K := 2;
   L := 3;
   E1(1) := 1.0;
   E1(2) := 2.0;
   E1(3) := 3.0;
   for I in 1...N9 loop
     P0;
   end loop;
```

```
-- end Module 9
         -- Module 10 : integer arithmetic
            J := 2;
            K := 3;
            for I in 1...N10 loop
               J := J + K;
               K := K + J;
               ] := K - ];
               K := K - J - J;
            end loop:
         -- end Module 10
         -- Module 11 : performing computations using standard
         --
                       mathematical functions
           X := 0.75;
            for I in 1..N11 loop
              X := SQRT(EXP(LN(X)/T1));
            end loop;
         -- end Moudle 11
     end loop CYCLE_LOOP;
     STOP_TIME := FLOAT(CPU_TIME_CLOCK); --Get Whetstone stop time
end WHETSTONE;
procedure COMPUTE_WHETSTONE_KIPS is
   -- Variables used to control execution of benchmark and to
   -- compute the Whetstone rating :
     NO_OF_RUNS : constant := 5; -- Number of times the benchmark is executed
     NO_OF_CYCLES : INTEGER; -- Number of times the group of benchmark
                              -- modules is executed
     -- I : INTEGER;
                  -- Embedded (as 10) in "N" constants at beginning of WHETSTONE proc
         -- Factor weighting number of times each module loops
         -- A value of ten gives a total weight for modules of
         -- approximately one million Whetstone instructions
     START_TIME : FLOAT;
                 -- Time at which execution of benchmark modules begins
     STOP_TIME : FLOAT;
                -- Time at which execution of benchmark modules ends
                -- (time for NO_OF_CYCLES)
     ELAPSED_TIME : FLOAT;
                  -- Time between START_TIME and STOP_TIME
     MEAN_TIME : FLOAT; -- Average time per cycle
RATING : FLOAT; -- Thousands of Whetstone instructions per sec
     MEAN_RATING : FLOAT;
                            -- Average Whetstone rating
     INT_RATING : INTEGER;
                             -- Integer value of KWIPS
     begin
       NEW_LINE;
       PUT_LINE
       ("Test Name: A000092
                                                    Class Name: composite");
       MEAN_TIME := 0.0;
       MEAN_RATING := 0.0;
       NO_OF_CYCLES := 10;
       RUN_LOOP:
       for RUN_NO in 1...NO_OF_RUNS loop
          -- Call the Whetstone benchmark parocedure
           WHETSTONE(NO_OF_CYCLES, START_TIME, STOP_TIME);
           -- Compute and write elapsed time
          ELAPSED_TIME := STOP_TIME - START_TIME;
           -- Sum time in milliseconds per cycle
          MEAN_TIME := MEAN_TIME + (ELAPSED_TIME*1000.0)/
                       FLOAT(NO_OF_CYCLES);
           -- Calculate the Whetstone rating based on the time for
           -- the number of cycles just executed and write
```

```
RATING := (1000.0 * FLOAT(NO_OF_CYCLES))/ELAPSED_TIME;
      -- Sum Whetstone rating
      MEAN_RATING := MEAN_RATING + RATING;
      INT_RATING := INTEGER(RATING);
      -- Reset NO_OF_CYCLES for next run using ten cycles more
      NO_OF_CYCLES := NO_OF_CYCLES + 10;
   end loop RUN_LOOP;
    -- Compute average time in millieseconds per cycle and write
   MEAN_TIME := MEAN_TIME/FLOAT(NO_OF_RUNS);
   NEW_LINE; PUT("Average time per cycle : ");
   PUT(MEAN_TIME,5,2,0); PUT_LINE(" milliseconds");
   -- Calculate average Whetstone rating and write
MEAN_RATING := MEAN_RATING/FLOAT(NO_OF_RUNS);
   INT_RATING := INTEGER(MEAN_RATING);
   NEW_LINE; PUT("Average Whetstone rating : ");
   PUT_LINE(INTEGER'IMAGE(INT_RATING) & " KWIPS");
   NEW_LINE;
   NEW_LINE;
end COMPUTE_WHETSTONE_KIPS;
begin
   COMPUTE_WHETSTONE_KIPS;
end A000092A;
```

```
80
-- PERFORMANCE MEASUREMENT : task creation and termination time
                             1 task no entry
---
- ---
                             task type in package, no select
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ;
package CREATE_PACK_1 is
  task type T1 is
 end T1 ;
 procedure P1 ; -- will create task, run task, and terminate task
end CREATE_PACK_1 ;
with CREATE_PACK_1 ; use CREATE_PACK_1 ;
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ; -- control optimization
with ITERATION ; -- obtain stable measurement
with PIWG_IO ; -- output results
procedure C000001 is -- main procedure to execute
  CPU_TIME : DURATION ; -- CPU time for one feature execution
  WALL_TIME : DURATION ; -- WALL time for one feature execution
  CHECK_TIMES : constant := 100 ; -- inside loop count and check
  ITERATION_COUNT : ITERATION.ITERATION_COUNTS ; -- set and varied by ITERATION package
  STABLE : BOOLEAN ; -- true when measurement stable
begin
  ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
  delay 5.0 ; -- wait for stable enviornment on some machines
  ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, ITERATION_COUNT increases each time
-- Control loop
----
    ITERATION.START_CONTROL ;
    for J in 1 .. ITERATION_COUNT loop
      GLOBAL := 0;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        GLOBAL := GLOBAL + A_ONE ;
        REMOTE :
      end loop ;
    end loop ;
    ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
- Test loop
-- establish task create and terminate time
    ITERATION.START_TEST ;
    for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
       P1 ; -- this has task that has global increment and call inside
      end loop ;
    end loop ;
    ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
    ITERATION.TEST_STABLE ( ITERATION_COUNT , STABLE ) ;
   exit when STABLE :
  end loop ;
  ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
```

```
- -
-- Printout
---
 " with one task, no entries, when task is in a procedure",
" using a task type in a package, no select statement, no loop, " );
end (000001;
package body CREATE_PACK_1 is
  task body T1 is
  begin
    GLOBAL := GLOBAL + A_ONE ;
    REMOTE ;
  end T1;
  procedure P1 is
   T : T1 ; -- this creates the task, runs task to completion and terminates
  begin
    null;
  end P1 ;
end CREATE_PACK_1 ;
```

```
82
-- PERFORMANCE MEASUREMENT : task creation and termination time
- -
                            1 task no entry
- -
                            task defined and used in procedure, no select
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ;
package CREATE_PACK_2 is
 procedure P1 ; -- will create task, run task, and terminate task
end CREATE_PACK_2 ;
with CREATE_PACK_2 ; use CREATE_PACK_2 ;
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ; -- control optimization
with ITERATION ; -- obtain stable measurement
with PIWG_IO ; -- output results
procedure COOOOO2 is -- main procedure to execute
 CPU_TIME : DURATION ; -- CPU time for one feature execution
 WALL_TIME : DURATION ; -- WALL time for one feature execution
 CHECK_TIMES : constant := 100 ; -- inside loop count and check
 ITERATION_COUNT : ITERATION.ITERATION_COUNTS ; -- set and varied by ITERATION package
  STABLE : BOOLEAN ; -- true when measurement stable
begin
  ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
 delay 0.5 ; -- wait for stable enviornment on some machines
 ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, ITERATION_COUNT increases each time
- -
-- Control loop
- -
   ITERATION.START_CONTROL ;
   for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
       GLOBAL := GLOBAL + A_ONE ;
       REMOTE ;
     end loop ;
    end loop ;
    ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
-- Test loop
    ITERATION.START_TEST ;
    for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0;
     for INSIDE_LOOP in 1 .. CHECK_TIMES loop
       P1 ; -- this has task that has global increment and call inside
     end loop ;
   end loop ;
   ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
    ITERATION.TEST_STABLE ( ITERATION_COUNT , STABLE ) ;
   exit when STABLE ;
 end loop ;
 ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
- -
-- Printout
 " Task create and terminate time measurement.
    " with one task, no entries when task is in a procedure,",
   " task defined and used in procedure, no select statement, no loop " ) ;
```

end (000002;

```
package body CREATE_PACK_2 is
 procedure P1 is
            this creates the task, runs task to completion and terminates execution time for task taken out by control loop
--'
- -
    task T1 is
    end T1 ;
    task body T1 is
    begin
      GLOBAL := GLOBAL + A_ONE ;
      REMOTE ;
    end T1 ;
  begin
   null;
  end P1;
end CREATE_PACK_2 ;
```

```
-- PERFORMANCE MEASUREMENT : operations on boolean arrays
- -
                             arrays are NOT packed
- -
                             operations on components in loop
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ; -- control optimization
with ITERATION ;
                                         -- obtain stable measurement
with PIWG_IO ;
                                         -- output results
procedure H000004 is -- main procedure to execute
  CPU_TIME : DURATION ;
                                   -- CPU time for one feature execution
  WALL_TIME : DURATION ;
                                   -- WALL time for one feature execution
  CHECK_TIMES : constant := 100 ; -- inside loop count and check
  ITERATION_COUNT : ITERATION.ITERATION_COUNTS ; -- set and varied by ITERATION package
  STABLE : BOOLEAN ;
                                   -- true when measurement stable
-- Boolean array declarations
  type UNPACKED_BIT_ARRAY is array ( NATURAL range \diamond ) of BOOLEAN;
  BIT_VALUE_1 : BOOLEAN := GLOBAL > 0;
  BIT_VALUE_2 : BOOLEAN := GLOBAL rem 2 = 0;
  BIT_VALUE_3 : BOOLEAN := GLOBAL <= 1;</pre>
  subtype UNPACKED_16 is UNPACKED_BIT_ARRAY ( 0 .. 15 );
  UNPACKED_1 : UNPACKED_16 := UNPACKED_16'( 0|3|6|9|12|15 => BIT_VALUE_1,
                                            11517111113 => BIT_VALUE_2,
                                            others => BIT_VALUE_3 );
  UNPACKED_2 : UNPACKED_16 := UNPACKED_16'( 0..3 => BIT_VALUE_1,
                                             4..12 => BIT_VALUE_2
                                            others => BIT_VALUE_3 );
begin -- procedure H000004
  ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
  delay 0.5 ; -- wait for stable enviornment on some machines
  ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, ITERATION_COUNT increases each time
-- Control loop
    ITERATION.START_CONTROL ;
    for J in 1 .. ITERATION_COUNT loop
      GLOBAL := 0
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        GLOBAL := GLOBAL + A_ONE ;
        REMOTE ;
      end loop ;
    end loop ;
    ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
- -
-- Test loop
    ITERATION.START_TEST ;
    for J in 1 .. ITERATION_COUNT loop
      GLOBA1 := 0 :
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        GLOBAL := GLOBAL + A_ONE;
        for I in UNPACKED_16'RANGE loop
          UNPACKED_1( I ) := UNPACKED_2( I ) xor not UNPACKED_1( I );
        end loop;
        for I in UNPACKED_16'RANGE loop
         UNPACKED_2( I ) := UNPACKED_1( I ) or UNPACKED_2( I );
        end loop;
        for I in UNPACKED_16'RANGE loop
          UNPACKED_1( I ) := not( UNPACKED_1( I ) and UNPACKED 2( I ) );
        end loop;
        REMOTE:
      end loop ;
```

```
end loop ; ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
---
-- Be sure UNPACKED_1 has been computed
- -
   if UNPACKED_1( GLOBAL rem 16 ) then
     GLOBAL := A_ONE;
     REMOTE;
   end if;
   ITERATION.TEST_STABLE ( ITERATION_COUNT , STABLE ) ;
   exit when STABLE ;
 end loop ;
 ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
- -
-- Printout
 " For this test the arrays are NOT PACKED with the pragma 'PACK.'" ,
   " For this test the operations are performed on components in a loop." ) ;
```

```
end H000004 ;
```

```
86
```

```
-- PERFORMANCE MEASUREMENT : Minimum procedure call and return time
- -
                            procedure local
- -
                            no parameters
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ;
with ITERATION ;
with PIWG_IO ;
procedure P000001 is -- main procedure to execute
  CPU_TIME : DURATION ;
  WALL_TIME : DURATION ;
  CHECK_TIMES : constant := 100 ;
  ITERATION_COUNT : ITERATION.ITERATION_COUNTS ;
  ITS_OK : BOOLEAN ;
  procedure PROC_0 is -- may be inlined thus zero time
  beain
    GLOBAL := GLOBAL + A_ONE ;
    REMOTE ;
  end;
begin
  ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
  delay 0.5 ; -- wait for stable enviornment on some machines
  ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, ITERATION_COUNT increases each time
-- Control loop
    ITERATION.START_CONTROL ;
    for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
       GLOBAL := GLOBAL + A_ONE ;
       REMOTE ;
     end loop ;
    end loop ;
    ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
-----
-- Test loop
    ITERATION.START_TEST ;
    for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0 ;
     for INSIDE_LOOP in 1 .. CHECK_TIMES loop
       PROC_0 ; -- this has control global increment and call inside
     end loop ;
   end loop ;
    ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
   ITERATION.TEST_STABLE ( ITERATION_COUNT , ITS_OK ) ;
   exit when ITS_OK ;
 end loop ;
 ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
- -
-- Printout
- -
 " Procedure call and return time ( may be zero if automatic inlining ) " ,
     " procedure is local " ,
     " no parameters " );
end P000001 ;
```

```
-- PERFORMANCE MEASUREMENT : procedure call and return time
-- procedure in package
```

```
- -
                             procedure in package
- -
                             ten discrete "in" parameters
package PROC_PACKAGE_10 is
 procedure PROC_0 ( A1, A2, A3, A4, A5, A6, A7, A8, A9, A10 : in INTEGER );
end PROC_PACKAGE_10 ;
with PROC_PACKAGE_10 ; use PROC_PACKAGE_10 ;
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ; -- control optimization
with IFERATION : obtain stable measurement
with PIWG_IO ; -- output results
procedure P000010 is -- main procedure to execute
  CPU_TIME : DURATION ; -- CPU time for one feature execution
  WALL_TIME : DURATION ; -- WALL time for one feature execution
  CHECK_TIMES : constant := 100 ; -- inside loop count and check
  ITERATION_COUNT : ITERATION.ITERATION_COUNTS ; -- set and varied by ITERATION package
 STABLE : BOOLEAN ; -- true when measurement stable
 A1 : INTEGER := A_ONE ;
 A2 : INTEGER := A1 + A_ONE
 A3 : INTEGER := A2 + A_ONE
 A4 : INTEGER := A3 + A_ONE
 A5 : INTEGER := A4 + A_ONE
 A6 : INTEGER := A5 + A_ONE
 A7 : INTEGER := A6 + A_ONE
 A8 : INTEGER := A7 + A_ONE ;
 A9 : INTEGER := A8 + A_ONE ;
 A10 : INTEGER := A9 + A_ONE ;
begin
 ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
 delay 0.5 ; -- wait for stable enviornment on some machines
  ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, 1/ERATION_COUNT increases each time
-- Control loop
. .
    ITERATION.START_CONTROL ;
   for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0 ;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        GLOBAL := GLOBAL + A1+A2+A3+A4+A5+A6+A7-A8-A9-A10 ;
       REMOTE ;
     end loop ;
   end loop ;
   ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
-- Test loop
    ITERATION.START_TEST ;
    for J in 1 .. ITERATION_COUNT loop
     GLOBAL := 0;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
       PROC_0 ( A1, A2, A3, A4, A5, A6, A7, A8, A9, A10 );
        -- this has control global increment and call inside
     end loop ;
    end loop ;
    ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
    ITERATION.TEST_STABLE ( ITERATION_COUNT , STABLE ) ;
   exit when STABLE ;
 end loop ;
 ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
```

```
-- Printout

--

PIWG_IO.PIWG_OUTPUT ( "P000010" , "Procedure" ,

CPU_TIME , WALL_TIME , ITERATION_COUNT ,

" Procedure call and return time measurement" ,

" Compare to P000005 " ,

" 10 parameters, in INTEGER " ) ;

end P000010 ;
```

```
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ;
package body PROC_PACKAGE_10 is -- compare to P000005
procedure PROC_0 ( A1, A2, A3, A4, A5, A6, A7, A8, A9, A10 : in INTEGER ) is
begin
    GLOBAL := GLOBAL + A1+A2+A3+A4+A5+A6+A7-A8-A9-A10 ;
    REMOTE ;
end ;
end ;
end PROC_PACKAGE_10 ;
```

```
-- PERFORMANCE MEASUREMENT : Minimum entry call and return time
- -
                             task inside procedure
                             1 task 1 entry
--
---
                             no select, do..end
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ; -- control optimization
with ITERATION ; -- obtain stable measurement
with PIWG_IO ; -- output results
procedure T000001 is -- main procedure to execute
  CPU_TIME : DURATION ; -- CPU time for one feature execution
  WALL_TIME : DURATION ; -- WALL time for one feature execution
  CHECK_TIMES : constant := 100 ; -- inside loop count and check
  ITERATION_COUNT : ITERATION.ITERATION_COUNTS ; -- set and varied by ITERATION package
  STABLE : BOOLEAN ; -- true when measurement stable
  task T1 is
   entry E1;
  end T1;
  task body T1 is
  begin
    loop
      accept E1 do
        GLOBAL := GLOBAL + A_ONE ;
        REMOTE :
      end E1;
    end loop ;
  end;
begin
  ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
  delay 00.5 ; -- wait for stable enviornment on some machines
  ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, ITERATION_COUNT increases each time
--
-- Control loop
    ITERATION.START_CONTROL ;
    for J in 1 .. ITERATION_COUNT loop
      GLOBAL := 0:
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        GLOBAL := GLOBAL + A_ONE ;
        REMOTE :
     end loop ;
    end loop ;
    ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
-- Test loop
---
    ITERATION.START_TEST ;
    for J in 1 .. ITERATION_COUNT loop
      GLOBAL := 0
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        T1.E1 ; -- this has control global increment and call inside
      end loop ;
    end loop ;
    ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
    ITERATION.TEST_STABLE ( ITERATION_COUNT , STABLE ) ;
    exit when STABLE ;
 end loop ;
```

```
ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
```

abort T1 ;

end T000001 ;

```
91
```

```
-- PERFORMANCE MEASUREMENT : tasks entry call and return time
                              1 task 2 entries
- -
---
                              one select statement
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ;
package TASK_PACK_4 is
  task T1 is
    entry E1 ;
    entry E2 ;
  end T1 ;
end TASK_PACK_4 ;
with TASK_PACK_4 ; use TASK_PACK_4 ;
with REMOTE_GLOBAL ; use REMOTE_GLOBAL ; -- control optimization
with ITERATION ; -- obtain stable measurement
with PIWG_IO ; -- output results
procedure T000004 is -- main procedure to execute
  CPU_TIME : DURATION ; -- CPU time for one feature execution
  WALL_TIME : DURATION ; -- WALL time for one feature execution
  CHECK_TIMES : constant := 100 ; -- inside loop count and check
  ITERATION_COUNT : ITERATION.ITERATION_COUNTS ; -- set and varied by ITERATION package
  STABLE : BOOLEAN ; -- true when measurement stable
  CASE_COUNT : constant := 2 ;
begin
  ITERATION.START_CONTROL ; -- dummy to bring in pages on some machines
  delay 0.5 ; -- wait for stable enviornment on some machines
  ITERATION.INITIALIZE ( ITERATION_COUNT ) ;
  loop -- until stable measurement, ITERATION_COUNT increases each time
_ - -
-- Control loop
    ITERATION.START_CONTROL ;
    for J in 1 .. ITERATION_COUNT loop
      GLOBAL := 0;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        GLOBAL := GLOBAL + A_ONE ;
        REMOTE :
      end loop ;
    end loop ;
    ITERATION.STOP_CONTROL ( GLOBAL , CHECK_TIMES ) ;
-- Test loop
    ITERATION.START_TEST;
    for J in 1 .. ITERATION_COUNT loop
      GLOBAL := 0;
      for INSIDE_LOOP in 1 .. CHECK_TIMES loop
        T1.E1 ; -- this has control global increment and call inside
        T1.E2 ; -- this has control global increment and call inside
      end loop ;
    end loop ;
    GLOBAL := GLOBAL / CASE_COUNT ;
    ITERATION.STOP_TEST ( GLOBAL , CHECK_TIMES ) ;
ITERATION.TEST_STABLE ( ITERATION_COUNT , STABLE ) ;
    exit when STABLE ;
  end loop ;
  ITERATION.FEATURE_TIMES ( CPU_TIME , WALL_TIME ) ;
  CPU_TIME := DURATION ( CPU_TIME / CASE_COUNT )
  WALL_TIME := DURATION ( WALL_TIME / CASE_COUNT ) ;
```

```
--
-- Printout
-----
  PIWG_IO.PIWG_OUTPUT ( "T000004" , "Tasking" ,
CPU_TIME , WALL_TIME , ITERATION_COUNT ,
" Task entry call and return time measured" ,
      " One tasks active, two entries, tasks in a package " ,
      " using select statement " );
  abort T1 ;
end T000004 ;
package body TASK_PACK_4 is
  task body T1 is
  begin
     loop
       select
          accept E1 do
            GLOBAL := GLOBAL + A_ONE ;
REMOTE ;
          end E1;
       or
          accept E2 do
            GLOBAL := GLOBAL + A_ONE ;
            REMOTE ;
          end E2 ;
       end select;
     end loop ;
  end T1;
end TASK_PACK_4 ;
```

Appendix C

Selected DIANA Representations of Test Programs

The timed fragment from P000001 and the entirety of SimpleTasks is included here in DIANA form. This DIANA form is that used on the Rational machine. It represents as a bracketed list with a node type tag, nonstructural attributes, and child nodes (structural attributes) in that order, e.g., [dn_type attr1 attr2 [child1] [child2]]. The hexadecimal numbers to the left are memory addresses for the nodes and can be ignored. Semantic attributes of the form sm_attr = [dn_tag ^] represent a pointer to a specific existing node of the type indicated.

POOOO01:

1FC910A_10C7B:	[DN_LOOP
	$lx_line_count = 7$
1FC910A_10E82:	[DN_FOR
1FC910A_10EFB:	[DN_ITERATION_ID
	$SM_SEQNUM = 1$
	SM_PARENT = [DN_PROC_ID ^]
	lx_symrep = "J"
	$sm_obi_type = [DN_RANGE ^]$
]
1FC910A_1101B;	TON_RANGE
	$sm_base_type = [DN_INTEGER ^]$
1FC910A_110B7:	TDN_NUMERIC_LITERAL
	lx_numrep = "1"
	$sm_exp_type = [DN_INTEGER ^]$
	sm_value = 1
	1
1FC910A_11136:	FDN_USED_OBJECT_ID
	lx_symrep = "ITERATION_COUNT"
	<pre>sm_defn = [DN_VAR_ID ^]</pre>
	$sm_exp_type = [DN_CONSTRAINED ^]$
	sm_value = No value
	1
	1
	1
1FC910A_11FE1:	[DN_STM_S
	$lx_line_count = 5$

	94
1FC910A_111F3:	[DN_ASSIGN
1FC910A_1127D:	[DN_USED_OBJECT_ID
	lx_symrep = "GLOBAL" sm_defn = [DN_VAR_ID ^]
	<pre>sm_exp_type = [DN_CONSTRAINED ^] sm_value = Unipitialized</pre>
1FC910A_1131F:	[DN_NUMERIC_LITERAL lx_numrep = "0"
	<pre>sm_exp_type = [DN_INTEGER ^] sm_value = @</pre>
1FC910A 113B9:] FDN_LOOP
	$lx_line_count = 4$
1FC910A_115C0: 1FC910A_11639:	[DN_FOR [DN_ITERATION_ID
	SM_SEQNUM = 1 SM_PARENT = [DN_PROC_ID_^]
	lx_symrep = "INSIDE_LOOP"
	sm_odj_type = [UN_KANGC ^]]
1FC910A_11759:	[DN_RANGE Sm base type = [DN_INTEGER ^]
1FC910A_117F5:	[DN_NUMERIC_LITERAL
	lx_numrep ≠ "1" sm_exp_type = [DN_INTEGER ^]
	sm_value = 1
1FC910A_11874:	[DN_USED_OBJECT_ID
	lx_symrep = "CHECK_IIMES" sm_defn = FDN_NUMBER_ID ^]
	<pre>sm_exp_type = [DN_INTEGER ^]</pre>
	Sm_vulue = 100
	ן ו
1FC910A_11F71:	[DN_STM_S
1FC910A_11931:	[DN_ASSIGN
1FC910A_119BB:	lx_line_count = 1 [DN_USED_OBJECT_ID
	lx_symrep = "GLOBAL"
	$sm_exp_type = [DN_CONSTRAINED ^]$
	sm_value = Uninitialized]
1FC910A_11A5D:	[DN_FUNCTION_CALL
	sm_exp_type = [bm_lnredek ~] sm_value = Uninitialized
	sm_normalized_param_s = [DN_EXP_S ^] lx_prefix = FALSE
1FC910A_11B1C:	[DN_USED_BLTN_OP
	SM_UKIGINAL_HOUE = [DH_USED_UP //] lx_symrep = "+"
	sm_operator = INTEGER_ADD 1
1FC910A_11D7D:	DN_PARAM_ASSOC_S
1+CA109T11BCL:	lx_symrep = "GLOBAL"
	sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^]
	sm_value = Uninitialized
1FC910A_11CA6:	J [DN_USED_OBJECT_ID
	lx_symrep = "A_ONE" sm defn _ FDN VAR TD Al
	<pre>sm_ccern = [ch_thi_ld] sm_exp_type = [DN_CONSTRAINED ^]</pre>
	sm_value ≠ Uninitialized]
	_] ⁻
	ſ

1FC910A_11DF7: 1FC910A_11E9B:

1FC910A_11F12:

] [DN_PROCEDURE_CALL _rRUCEDURE_CALL
lx_line_count = 1
sm_normalized_param_s = [DN_EXP_S ^]
[DN_USED_NAME_ID
lx_symrep = "REMOTE"
sm_defn = [DN_PROC_ID ^]] [DN_PARAM_ASSOC_S]]]

95

]

ן ז

SimpleTasks:

1F77D0A_AD12:	[DN_COMP_UNIT
	IX_line_count = 55 SM ID TABLE =
1F77D0A_CE56:	[DN_CONTEXT
	lx_line_count = 0
1677D94 (FC6.	J FON SURDROCRAM RODY
TF//DUA_CECO.	SM_FORWARD = [DN_SUBPROGRAM_DECL ^]
	<pre>lx_line_count = 55</pre>
1F77D0A_CF6A:	[DN_PROC_ID
	lx symmet = "DOIT"
	sm_spec = [DN_PROCEDURE ^]
	sm_body = [DN_BLOCK ^]
	sm_scub = null sm first = FDN PROC ID ^]
]
1F77D0A_D0AC:	[DN_PROCEDURE
1F//00A_0105.	lx_line_count = 0
	SM_ID_TABLE =
1F77D0A_D19B:	
	POST_COMMENT_HEIGHT = -2
	$PRE_COMMENT_HEIGHT = -2$
1F77D0A_13D74:	TON_ITEM_S
	$lx_line_count = 49$
1F7700A_025A:	LDN_VAR
1F77D0A_D554:	[DN_ID_S
1F77D0A_D2FE:	[DN_VAR_ID
	SM_PARENI = [UN_PROL_ID ^]
	sm_obj_type = [DN_CONSTRAINED ^]
	<pre>sm_obj_def = [DN_NUMERIC_LITERAL ^]</pre>
1F77D04 0429	J FDN VAR TO
1.1.1.000.20.20.	SM_PARENT = [DN_PROC_ID ^]
	lx_symrep = "COUNT_B"
	sm_obj_type = [UN_CONSIRAINED ^] sm_obj_def = [DN_NIMFETC_ITTERAL_A]
1577004 0580.	
1F7700A_0363:	SWALNED SWALNED FOR INTEGER AT
	<pre>sm_base_type = [DN_INTEGER ^]</pre>
1577004 0605	Sm_constraint = [DN_RANGE ^]
1F7700A_0093.	lx symrep = "INTEGER"
	<pre>sm_defn = [DN_TYPE_ID ^]</pre>
	Ι Γρη γοτρι
]
1F77D0A_D70C:	[DN_NUMERIC_LITERAL
	sm exp type = 10 INTEGER ^]
	sm_value = 0
	,]
1F77D0A_D7A6:	DN_VAR
	POST_COMMENT_HEIGHT = -2
1F7700A D975:	f_{DN} ID S
1F77D0A_D84A:	[DN_VAR_ID
	SM_PARENT = [DN_PROC_ID ^]
	sm_obj_type = [DN_CONSTRAINED ^]
	<pre>sm_obj_def = [DN_NUMERIC_LITERAL ^]</pre>
	, []]
1F77D0A_D9D4:	DN_CONSTRAINED
	<pre>sm_type_struct = [DN_INTEGER ^]</pre>
	sm_base_type = [UN_INIEGER ^]
1F77DØA_DAB6:	[DN_USED_NAME_ID
	lx_symrep = "INTEGER"
	[DN_VOID]
1577004 0920.] FON NUMEDIC LITEDAL
TO TOWN DOED.	lx_numrep = "6"
	<pre>sm_exp_type = [DN_INTEGER ^]</pre>
	sm_value = 6
) ¹
1F77D0A_DC0A:	[DN_TASK_DECL
	POSI_COMMENT_HEIGHT # "2

1F77D&A_DC94:	<pre>lx_line_count = 3 [OM_VAR_ID SM_PARENT = [DM_PROC_ID ^] lx_symrep = "TASK_A" sm_obj_type = [OM_TASK_SPEC ^] sm_obj_def = [OM_TASK_SPEC ^]</pre>
1F77D0A_DD8A:] [DN_TASK_SPEC]x line count = 0
1F77D0A_DE1D:	Sm_body = [DN_BLOCK ^] [DN_DECL_S LX_VERBOSE = TRUE lx_line_count = 0 SM_ID_TABLE =
	_ ت _ ۱
1F77D8A_DF3F:] [DN_TASK_DECL POST_COMMENT_HEIGHT = -2
1F77D@A_DFC9:	<pre>lx_line_count = 3 [DM_VAR_ID SM_PARENT = [DN_PROC_ID ^] lx_symrep = TASK_B" sm_obj_type = [DN_TASK_SPEC ^] count = count = [DN_TASK_SPEC ^] </pre>
1F77D0A_E08F:	Sm_ooj_det = [UM_IASK_SPEC ^]] [DM_TASK_SPEC]x light = 0
1F77D0A_E152:	sm_body = [D_BLOCK ^] [DN_DECL_S LX_VER80SE = TRUE
	lx_line_count = 0 SM_ID_TABLE =]
	1
1F77D0A_E274:	[DN_TASK_DECL POST_COMMENT_HEIGHT = -2
1F77D0A_E2FE:	[DN_VAR_ID SM_PARENT = [DN_PROC_ID ^]
	sm_obj_type = [DN_TASK_SPEC ^] sm_obj_def = [DN_TASK_SPEC ^]
1F77D0A_E3F4:] [DN_TASK_SPEC lx_line_count = 0
1F77D@A_EDC0:	sm_body = [DN_BLOCK ^] [DN_DECL_S
1F77D0A_E487:	lx_line_count = 3 [DN_SUBPROGRAM_DECL
1F77D0A_E52B:	lx_line_count = 1 [DN_ENTRY_ID SM_SEONUM = 1
	SM_PARENT = [DN_VAR_ID ^] lx_symrep = "ENTRY_A" sm spec = [DN_ENTRY_^]
1F77D8A_E64B:] [DN_ENTRY lx line count = 0
1F77D0A_E6D5:	[DN_VOID] [DN_PARAM_S
	SM_ID_TABLE =
1F77D0A_E79A:	DN_SUBPROGRAM_DECL
1F77D0A_E83E:	
	SM_PARENT = [DN_VAR_ID ^] [x_symrep = "ENTRY_B" sm_spec = [DN_ENTRY ^]
1F77D0A_E95E:] [DN_ENTRY 1x line count = 0
1F77D0A_E9E8:	[DW_VOID] [DW_PARAM_S lx_line_count = 0 SM_ID_TABLE =
	ן [סא_ void]
1F77D0A_EAAD:] [DN_SUBPROGRAM_DECL
- 1F77DØA_EB51:	1x_1ine_count = 1 [DN_ENTRY_ID
	SM_SEQNUM = 3 SM_PARENT = [DN_VAR_ID ^] lx_symred = "DONE"
	sm_spec = [DN_ENTRY ^]]

	98
1F77D9A_EC71:	[DN_ENTRY]x line count = 0
1F77D0A_FCFB:	[DN_VOID] [DN_PARAM_S lx_line_count = 0 SM ID TABLE =
	1 1
	[DN_V0ID]
	,] ⁻
1F77D8A_F14A:] [DN_TASK_BOOY SM_FORMARD = [DN_TASK_DECL ^]
1F77D0A_F1D4:	[DN_TASK_BODY_ID] X symmetry _ TASK A"
	sm_spec = [DN_TASK_SPEC ^] sm_body = [DN_BLOCK ^] sm_stub = null sm_first = [DN_VAR_ID ^]
1F77D@A_F2C3:] [DN_BLOCK
	POST_COMMENT_HEIGHT = -2 lx_line_count = 8
1F77D0A_F367:	[DN_ITEM_S lx_line_count = 0
	SM_ID_TABLE =
1F77D8A_106E3:	[DN_STM_S lx_line_count = 6
1F77D0A_F412:	lum_loop lx_line_count = 3
1F77D0A_F692:	LUM_FUN [DN_ITERATION_ID SM_SECONUM 1
	SM_PARENT = [DN_VAR_ID ^] lx_symrep = "I" sm_odj_type = [DN_RANGE ^]
1F 77D9A_F 7B2 :] [DN_RANGE
1F77D8A_F84E:	<pre>sm_base_type = [DN_INTEGER ^] [DN_NUMERIC_LITERAL</pre>
	sm_value = 1
1F77DØA_F8CD:] [DN_NUMERIC_LITERAL
	SM_INT_VALUE = 1000 lx_numrep = "1000"
	sm_exp.type = [DN_INTEGER ^] sm_value = 1000
1F77D8A_FE62:	J [DN_STM_S
1F77D@A_F9DA:	
1F77D8A_FA64:	
	sm_cdefn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^]
	sm_value = Uninitialized
1F7/D@A_FB06:	[DN_FUNCTION_CALL sm_exp_type = [DN_INTEGER ^] sm_value = Uninitialized sm_normalized param.s = [DN_EXP_S ^]
1F77DØA_F8C5:	lx_prefix = FALSE [DN_USED_BLTN_OP
	SM_ORIGINAL_NODE = [DN_USED_OP ^] lx_symrep = "+"
	sm_operator = INIEGER_ADD
1F77D0A_FE03: 1F77D0A_FC78:	[UM_PARAM_ASSU_S [DM_USED_OBJECT_ID
	sm_exp_type = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^] sm_value = Uninitialized
1F77D0A_FD4F:	DN_NUMERIC_LITERAL
	sm_exp_type = 1 Sm_volue = 1
	_ 1
]
1F77D8A_FED2:	LON_ENTRY_CALL

SM_ORIGINAL_NODE = [DN_PROCEDURE_CALL ^] lx_line_count = 1
sm_normalized_param_s = [DN_EXP_S ^] 1F77D0A_FF76: [DN_SELECTED _stEtClEU sm_exp_type = null sm_value = Uninitialized [DN_USED_OBJECT_ID lk_symrep = "TASK_C" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_TASK_SPEC ^] sm_value = Uninitialized 1F77D0A_1001A: J [DN_USED_NAME_ID lx_symrep = "ENTRY_A" sm_defn = [DN_ENTRY_ID ^] 1F77D0A 100BC: 1 [DN_PARAM_ASSOC_S] 1F77D0A_10133: 1F77D0A_10238: DN_ASSIGN 1F77D0A_102C2: DN_FUNCTION_CALL 1F77D0A_10364: sm_exp_type = [DN_INTEGER ^] sm_value = Uninitialized sm_normalized_param_s = [DN_EXP_S ^] lx_prefix = FALSE
fDN_USED_BLTN_OP 1F77D0A_10423: SM_ORIGINAL_NODE = [DN_USED_OP ^] lx_symrep = "+" sm_operator = INTEGER_ADD DN_PARAM_ASSOC_S 1F77D0A_10684: 1F77D0A_104D6: _PARAM_ASSOC_S [DN_USED_OBJECT_ID lx_symrep = "COUNT_A" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^] sm_value = Uninitialized] [DN_USED_OBJECT_ID lx_symrep = "COUNT_A" sm_defn = [DN_VAR_ID ^] sm_exp_type = [ON_CONSTRAINED ^] sm_value = Uninitialized 1F77D0A_105AD:]] 3] 1F77D0A_10753: [DN_ALTERNATIVE_S lx_line_count = 0 1] [DN_TASK_BODY SM_FORWARD = [DN_TASK_DECL ^] 1F77D0A 10891: SM_FORMARD = [DN_TASK_DECL ^]
lx_line_count = 9
[DN_TASK_BODY_ID
 lx_symrep = "TASK_B"
 sm_spec = [DN_TASK_SPEC ^]
 sm_body = [DN_BLOCK ^]
 sm_stub = null
 sm_first = [DN_VAR_ID ^] 1F77D0A_1091B: 1F77D0A_10A0A: DN_BLOCK POST_COMMENT_HEIGHT = -2 lx_line_count = 8 1F77D0A_10AAE: [DN_ITEM_S lx_line_count = 0
SM_ID_TABLE =] [DN_STM_S lx_line_count = 6 1F77D0A_11DDA: 1F77D0A_10859: lx_line_count = 3 [DN_FOR 1F77D0A_10D60: 1F77D0A_10DD9: _FOR [DN_ITERATION_ID SM_SEQNUM = 1 SM_PARENT = [DN_VAR_ID ^] lx_symrep = "I" sm_obj_type = [DN_RANGE ^] 1F77D0A_10EF9: DN_RANGE MANGE sm_base_type = [DN_INTEGER ^] [DN_NUMERIC_LITERAL lx_numrep = "1" sm_exp_type = [DN_INTEGER ^] sm_exp_type 1F77D0A_10F95: sm value = 1 1
1F77D0A_11014:	100 [DN_NUMERIC_LITERAL lx_numrep = "100"
	sm_exp_type = [DN_INTEGER ^] sm_value = 100]]
1F7700A_11559:] [DN_STM_3
1F77D0A_110AE:	lx_line_count = 1 [DN_ASSIGN
1F77D0A_11138:	<pre>lx_line_count = 1 [DN_USED_OBJECT_ID lx_symrep = "COUNT_B" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^] sm_value = Uninitialized</pre>
1F77D0A_111DA:] [DN_FUNCTION_CALL sm_exp_type = [DN_INTEGER ^] sm_value = Uninitialized sm_normalized_param_s = [DN_EXP_S ^] lx_prefix = FALSE
1F77D0A_11299:	[DN_USED_BLTN_OP SM_ORIGINAL_NODE = [DN_USED_OP ^] lx_symrep = "+" sm_operator = INTEGER_ADD]
1F77D0A_114FA: 1F77D0A_1134C:	[DN_PARAM_ASSOC_S [DN_USED_OBJECT_ID lx_symrep = "COUNT_B" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^] sm_value = Uninitialized
1F77D0A_11423:	[DN_USED_OBJECT_ID lx_symrep = "I" sm_defn = [DM_ITERATION_ID ^] sm_exp_type = [DN_RANGE ^] sm_value = Uninitialized]
	ر د ر
1F77D0A_115C9:] [DN_ENTRY_CALL SM_ORIGINAL_NODE = [DN_PROCEDURE_CALL ^] lx_line_count = 1
1F77D0A_1166D:	<pre>sm_normalized_param_s = [DN_EXP_S ^] [DN_SELECTED sm_exp_type = null rep_type = listentialized</pre>
1F77D0A_11711:	Sm_vdtde = Vinit([][220 [DN_USED_08JECT_ID lx_symrep = "TASK_C" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_TASK_SPEC ^] sm_value = Uninitialized
1F77D0A_117B3:] [DN_USED_NAME_ID lx_symrep = "ENTRY_B" sm_defn = [DN_ENTRY_ID ^]]
1F77D0A_1182A:] [DN_PARAM_ASSOC_S]
1F77D0A_1192F:	[DN_ASSIGN
1F77D0A_11989:	[DN_USED_OBJECT_ID [Lx_symrep = "COUNT_B" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^] sm_value = Uninitialized
1F77D0A_11A58:] [DN_FUNCTION_CALL sm_exp_type = [DN_INTEGER ^] sm_value = Uninitialized sm_normalized_param_s = [DN_EXP_S ^]
1F77D0A_11B1A:	[X_PFETX = FALSE [DN_USED_BLTN_OP SM_ORIGINAL_NODE = [DN_USED_OP ^] lx_symrep = "+" sm_operator = INTEGER_ADD
1F77D0A_11D7B: 1F77D0A_11BCD:	JON_PARAM_ASSOC_S [DN_USED_OBJECT_ID lx_symrep = "COUNT_B" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^] sm_value = Uninitiolized
1F77D0A_11CA4:	[ON_USED_OBJECT_ID lx_symrep = "COUNT_B" sm_defn = [DN_VAR_ID ^] sm_exp_type = [DN_CONSTRAINED ^]

101 sm_value = Uninitialized 1]]] 1F77D0A_11E4A: [DN_ALTERNATIVE_S lx_line_count = 0] 3 J [DN_TASK_BODY SM_FORMARD = [DN_TASK_DECL ^] lx_line_count = 16 [DN_TASK_BODY_ID lx_symrep = "TASK_C" sm_body = [DN_BLOCK ^] sm_stub = null sm_first = [DN_VAR_ID ^] 1F77D0A_11F88: 1F77D0A_12012: 1F77D0A_12101: DN_BLOCK POST_COMMENT_HEIGHT = -2 lx_line_count = 15 [DN_ITEM_S 1F77D0A_121A5: lx_line_count = 0
SM_ID_TABLE =] [DN_STM_S lx_line_count = 13 [DN_LOOP lx_line_count = 1F77D0A_13BC6: 1F77D0A_12250: lx_line_count = 11 [DN_FOR [DN_ITERATION_ID 1F77D0A_12457: 1F77D0A_124D0: SM_SEQNUM = 1 SM_PARENT = [DN_VAR_ID ^] lx_symrep = "I" sm_obj_type = [DN_RANGE ^] 1F77D0A_125F0: [DN_RANGE sm_base_type = [DN_INTEGER ^]
[DN_NUMERIC_LITERAL 1F77D0A_1268C: lx_numrep = "1" sm_exp_type = [DN_INTEGER ^] sm_value = 1] [DN_NUMERIC_LITERAL lx_numrep = "2" sm_exp_type = [DN_INTEGER ^] sm_value = 2 1F77D0A_1270B: ٦]] [DN_STM_S lx_line_count = 9 [DN_SELECT lx_line_count = 9 [DN_SELECT_CLAUSE_S lx_line_count = 8 [DN_SELECT_CLAUSE lx_line_count [N_SELECT_CLAUSE lx_line_count [N_SELECT_CLAUSE] 1F77D0A_13920: 1F77D0A_1278A: 1F77D0A_1380B: 1F77D0A_1282F: lx_line_count = 3
[DN_VOID]
[DN_STM_S
lx_line_count = 3
[DN_ACCEPT 1F77D0A_12F92: 1F77D0A_128D4: $lx_line_count = 3$ [DN_USED_NAME_ID lx_symrep = "ENTRY_A" sm_defn = [DN_ENTRY_ID ^] 1F77D0A_12978: DN_PARAM_S 1F77D0A_129EF: lx_line_count = 0 SM_ID_TABLE = [DN_STM_S lx_line_count = 1 [DN_ASSIGN 1F77D0A_12F22: 1F77D0A_12A9A: $lx_line_count = 1$ Intercont = 1
[DN_USED_OBJECT_ID
lx_symrep = "RESULT_C"
sm_defn = [DN_VAR_ID ^]
sm_exp_type = [DN_CONSTRAINED ^]
sm_value = Uninitialized 1F77D0A_12824:] [DN_FUNCTION_CALL sm_exp_type = [DN_INTEGER ^] sm_value = Uninitialized sm_normalized_param_s = [DN_EXP_S ^] lx_prefix = FALSE [DN_USED_BLTN_OP SM_OPICINAL_NOPE = [DN_USED_OP_A] 1F77D0A_128C6: 1F77D0A_12C85: SM_ORIGINAL_NODE = [DN_USED_OP ^] lx_symrep = "/" sm_operator = INTEGER_DIV 3



1F77D0A_13C36:] [DN_ALTERNATIVE_S lx_line_count = 0]
	, ,
1F77D0A_1463D:] [DN_STM_S PRE_COMMENT_HEIGHT = 1
1F77D0A_142F2:	[DN_ENTRY_CALL SM_ORIGINAL_NODE = [DN_PROCEDURE_CALL ^] lx line count = 1
1F77D0A_14396:	<pre>sm_normalized_param_s = [DN_EXP_S ^] [DN_SELECTED sm_exp_type = null</pre>
1F77D0A_1443A:	sm_value = Uninitialized [DN_USED_OBJECT_ID lx_symrep = "TASK_C" cm_defmCDN_VAP_ID_A]
1F77D0A_144DC:	sm_uern = [Um_TAK_SPEC ^] sm_exp_type = [ON_TASK_SPEC ^] sm_value = Uninitialized] [DN_USED_NAME_ID lx_symrep = "DONE" sm_defn = [DN_ENTRY_ID ^]]
1F77D0A_14553:	[DN_PARAM_ASSOC_S]
1F77D0A_146F0:] [DN_ALTERNATIVE_S lx_line_count = 0]
1F77D0A_1483C:] [DN_PRAGMA_S lx_line_count = 0]]

Appendix D

Experiment Output

main 1000 Iterations of simple tasking system takes	11.8167 seconds.
main	
1000 Iterations of simple tasking system takes	11.9333 seconds.
main 1000 Iterations of simple tasking system takes	11.9833 seconds.
main	
1000 Iterations of simple tasking system takes	12.0500 seconds.
main 1000 Itaustisus of simula teching system tolag	14 9994 accorda
1000 Iterations of simple tasking system takes	14.3834 seconds.
1000 Iterations of simple tasking system takes	14.3833 seconds.
main	
1000 Iterations of simple tasking system takes	14.4000 seconds.
main	1 (0000 1
1000 Iterations of simple tasking system takes	14.3833 seconds.
main	1 1 0 0 0 1
1000 Iterations of simple tasking system takes	14.3834 seconds.
main	4 4 9 9 9 9
1000 Iterations of simple tasking system takes	14.3833 seconds.
main	
1000 Iterations of simple tasking system takes	14.3834 seconds.
1000 Iterations of simple tasking system takes	11 7333 seconds
main	11.1000 5000145.
1000 Iterations of simple tasking system takes	14.5667 seconds.
main	
1000 Iterations of simple tasking system takes	14.5667 seconds.
main	1 5005
1000 Iterations of simple tasking system takes	14.5667 seconds.
main	1.1.5005
1000 Iterations of simple tasking system takes	14.5667 seconds.
main	4 × × 00 = 1
1000 Iterations of simple tasking system takes	14.5667 seconds.

1	
Test Name: A000090	Thomas in 1
Clock resolution measurement running	74557.0008 74557.0008 8.0080
Determine clock resolution using second differences	Test Iteration 1
of values returned by the function CPU Time Clock.	74558.0000 74558.0000 0.0000
	Iteration 2
Number of sample values is 7000	74558.0000 74558.0000 0.0000
Clock Resolution = 1.0000000000000 seconds.	1951 Iteration 2 74550 000 74559 0000 0 0000
Clock Resolution (average) = 1.000000000000 seconds	Teration A
	74559.0000 74559.0000 0.0000
	Test Iteration 4
Test Name: A000091 Class Name: Composite	74559.0000 74559.0000 0.0000
1.2000 is time in milliseconds for one Dhrystone	Iteration 8
Test Description:	74560.0000 74560.0000 0.0000
Reinhold P. Neicker's DHRYSTONE composite benchmark	Test Iteration 8
	74560.0000 74560.0000 0.0000
	1167A1100 10 74560 0000 74561 0000 1 0000
Test Name: AU00092 Class Rame: Composite	74360.0000 74361.0000 1.0000
Average time per cucle (2784-31 milligeconds	74561.0000 74562.0000 3.000
werage the bar dying i figering millingconde	Iteration 32
Average Whetstone rating : 359 KWIPS	74562.0000 74562.0000 0.0000
·······	Test Iteration 32
	74562.0000 74564.0000 2.0000
Iteration 1	Iteration 64
73717.0000 73717.0000 0.0000	_74564.0000 74565.0000 1.0000
Test Iteration 1	Test Iteration 54
	Territor 128
	74569-0000 74569-0000 0-0000
Test Iteration 2	Test Iteration 128
73718.0004 73718.0000 0.0000	74570.0000 74576.0000 6.0000
Iteration 4	Iteration 256
73718.0000 73718.0000 0.0000	74576.0000 74578.0000 2.0000
Test Iteration 4	1851 108781100 200 24578 0000 74501 0000 13 0000
73/17.0000 /3/17.0000 0.0000 Thomation 0	Tteration 512
73720.0000 73720.0000 0.0000	74591.0000 74594.0000 3.0000
Test Iteration 8	Test Iteration 512
73720.0000 73721.0000 1.0000	74594.0000 74620.0000 26.0000
Iteration 16	Iteration 1024
73721.0000 73721.0000 0.0000	74620.0000 74626.0000 6.0000
Test Iteration 16	Test Iteration 1024
73721.0000 73723.0000 2.0000	/46/6.0000 74678.0000 52.0000
Iteration J2	110781100 2048
Teat Iteration 32	Test Iteration 2048
73723.0000 73727.0000 4.0000	74690.0000 74793.0000 103.0000
Iteration 64	
73727.0000 73727.0000 0.0000	Test Name: H000004 Class Name: Chapter 13
Test Iteration 64	CPU Time: 449.2 microseconds
73727.0000 73734.0000 7.0000	Wall Time: 449.2 microseconds. Iteration Count: 2048
Iteration 128	Test Description:
73734.0000 73735.0000 1.0000	Time to perform standard boolean operations on arrays of booleans.
Tent (teration 128 73735 Anno 73746 Anno 14 Anno	For this test the arrays are NOT PACKED with the pragma 'PACK.'
73733.0000 /3/49.0000 14.0000	for this case the operations are performed on components in a roop.
73749,0000 73750,0000 1.0000	
Test Iteration 256	Iteration 1
73750.0000 73777.0000 27.0000	75295.0000 75295.0000 0.0000
Iteration 512	Test Iteration 1
73777.0000 73780.0000 3.0000	75296.0000 /5296.0000 0.0000
Test Iteration 51/	iteration 2
	75766 0000 75766 0000 0 0000
73780.0000 73834.0000 54.0000	75296.0000 75296.0000 0.0000 Test Iteration 2
73780.0000 73834.0000 54.0000 Iteration 1024 73834.0000 7383.0000 5.0000	75296.0000 75296.0000 0.0000 Test Iteration 2 75296.0000 75296.0000 0.0000
73780.000073894.0000 Terretion 1024 73834.000073839.0000 Teret Feyention 1024	75296.0000 75296.0000 0.0000 Teet Iteration 2 75296.0000 75296.0000 0.0000 Iteration 4
73780.000073834.0000 54.0000 Iteration 1024 73834.000073893.0000 5.0000 Test Iteration 1024 73839.000073947.000 108.0000	75296.0000 75296.0000 0.0000 Teest Titeration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000
73780.000073894.0000 54.0000 Iteration 1024 73834.000073839.0000 5.0000 Teet Iteration 1024 73839.000073947.0000 108.0000	75296.0000 75296.0000 0.0000 Teet Iteration 2 T5295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Teet Iteration 4
73780.000073894.0000 54.0000 Iteration 1024 73834.000073839.0000 5.0000 Temet Iteration 1024 73839.000073947.0000 Temet Mamen C000001 Class Mamer Tasking	75296.0000 75296.0000 0.0000 Teent Liceration 2 75296.0000 75296.0000 0.0000 Liceration 4 75297.0000 75297.0000 0.0000 Teet liceration 4 75297.0000 75297.0000 0.0000
73780.0000 73834.0000 54.0000 Iteration 1024 73334.0000 73839.0000 Test Iteration 1024 73339.0000 73834.0000 Test Iteration 1024 73339.0000 73894.0000 Test Iteration 1024 73839.0000 73897.0000 Class Name: Tasking CPU Time: 1005.9 microseconds 1005.9	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8
73780.0000 73834.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73834.0000 73837.0000 Test Hame: 10000 Test Hame: 1000.000 Test Hame: 1005.9 microseconds Wall Time: 1005.9 microseconds. Iteration Count: 1024	75296.0000 75296.0000 0.0000 Test Iteration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Test Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000
73780.0000 73834.0000 54.0000 Iteration 1024 73834.0000 78893.0000 73834.0000 738934.0000 5.0000 78893.0000 78874.0000 Test Iteration 1024 73839.0000 73874.0000 108.0000 Test Mame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Test Description: 1054	75296.0000 75296.0000 0.0000 Text teration 2 75295.0000 75296.0000 0.0000 Letation 4 75297.0000 75297.0000 0.0000 Text teration 4 75297.0000 75297.0000 0.0000 Letation 75297.0000 0.0000 75297.0000 75297.0000 0.0000
73780.0000 73892.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73830.0000 73837.0000 Test Iteration 1024 73830.0000 73947.0000 Test Name: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration: Class Name: Count: Test Description: Task create and terminate measurement	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000
T3780.0000 73934.0000 54.0000 Iteration 1024 7334.0000 73834.0000 5.0000 Test Iteration 1024 73378.0000 73837.0000 108.0000 Test Name: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Nall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement when task is in a procedure whin to me task, no entries, when task is in a procedure name: task town Lown	75296.0000 75296.0000 0.0000 Text Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 4 75297.0000 75297.0000 0.0000 75297.0000 75297.0000 0.0000 75297.0000 75297.0000 0.0000 75297.0000 75297.0000 0.0000 75297.0000 75297.0000 0.0000
73780.0000 73834.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 CPU Times 1005.9 microseconds Wall Times 1005.9 microseconds Wall Times 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop,	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000
73780.000073834.0000 54.0000 Iteration 1024 73834.000073839.0000 5.0000 Test Iteration 1024 73839.000073947.0000 108.0000 Test Mame: C000001 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop,	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Iteraticn 16 75298.0000 75298.0000 0.0000 Text Iteration 16 75298.0000 75298.0000 0.0000
73780.0000 54.0000 Iteration 1024 73834.0000 73834.0000 73839.0000 Test Learation 1024 73839.0000 73947.0000 Test Kame: C000001 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds With one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Test Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Test Iteration 16 75298.0000 75298.0000 0.0000 Test Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32
73780.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73834.0000 73837.0000 Test Name: 000001 CPU Time: 1005.9 Mail Time: 1005.9 Mail Time: 1005.9 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75298.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 12 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000
73780.0000 73834.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds With one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 73949.0000 0.0000	75296.0000 75296.0000 0.0000 Textication 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Textication 8 75297.0000 75297.0000 0.0000 Textication 8 75297.0000 75297.0000 0.0000 Textication 16 75298.0000 75298.0000 0.0000 Textication 32 75298.0000 75298.0000 0.0000 Textication 32 75298.0000 75298.0000 0.0000 Textication 32 75298.0000 75298.0000 0.0000
73780.0000 73834.0000 Iteration 1024 73839.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Names C000001 CPU Times 1005.9 microseconds Mall Times 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, 1 Iteration 1 73939.0000 73940.000 Test teration 1 73949.0000 0.0000	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73939.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds With one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 1 73939.0000 0.0000 Test Tiescation 1 1 73939.0000 1.0000 Iteration 2 0.000	75296.0000 75296.0000 0.0000 Textication 2 75295.0000 75296.0000 0.0000 Teratican 4 75297.0000 75297.0000 0.0000 Textican 8 75297.0000 75297.0000 0.0000 Textican 8 75297.0000 75297.0000 0.0000 Textican 16 75298.0000 75298.0000 0.0000 Textican 12 75298.0000 75298.0000 0.0000 Textican 32 75298.0000 75298.0000 0.0000 Textican 32 75298.0000 75298.0000 0.0000 Textican 32 75298.0000 75298.0000 0.0000 Textican 32 75298.0000 75298.0000 0.0000
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 Test Iterration 1024 CPU Times 1005.9 microseconds Wall Times 1005.9 microseconds Wall Times 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 0.0000 Test terration 1 73949.0000 0.0000 Test terration 1 1 73949.0000 1.0000 Terration 2 2	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 64
T3726.0000 73835.0000 54.0000 Iteration 1024 73834.0000 73835.0000 5.0000 Test Iteration 1024 73834.0000 7387.0000 108.0000 Test Xame: C00000 7387.0000 Test Xame: C00000 7387.0000 Test Xame: C00000 7387.0000 Test Xame: C000001 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 73949.0000 0.0000 Test Iteration 1 73949.0000 73950.0000 1.0000 Iteration 2 73950.0000 73950.0000 73950.0000 73950.0000 0.0000 Test Iteration 2 0.0000 Test Iteration 2 0.0000	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Teration 4 75297.0000 75297.0000 0.0000 Teration 75297.0000 0.0000 Teration 75297.0000 0.0000 Teration 7297.0000 75298.0000 75298.0000 Teration 22 75298.0000 75298.0000 Teration 32 75298.0000 75298.0000 Teration 32 75298.0000 75299.0000 Teration 64 75298.0000 75299.0000 75298.0000 75299.0000 Teration 64 75298.0000 75299.0000 Teration 64 75298.0000 75299.0000 Teration 64 75298.0000 75299.0000 Teration 64 75298.0000 75299.0000 Teration 64 75299.0000 75299.0000 Teration 64 Teration 64 Terat
73736.0000 534.0000 Terration 1024 73834.0000 73839.0000 Test Lerration 1024 73839.0000 73947.0000 Test Kamen C000001 CPU Timen 1005.9 microseconds Wall Timen 1005.9 microseconds Wall Timen 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, 1 Iteration 1 73939.0000 0.0000 Test teration 1 1 73949.0000 0.0000 Test teration 2 1 73950.0000 0.0000 Test teration 2 2 73950.0000 0.0000 Test teration 2 7 73950.0000 0.0000 Test teration 2 7 73950.0000 0.0000 Test teration 2 7 73950.0000 0.0000 Test teration 4 0.0000	75296.0000 75296.0000 0.0000 Text Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 42 75298.0000 75298.0000 0.0000 Iteration 64 75300.0000 75300.0000 0.0000
73780.000073952.0000 54.0000 Terration 1024 73834.000073947.0000 5.0000 Test Iteration 1024 73835.000073947.0000 108.0000 Test Name: C000001 CPU Time: 1005.9 microseconds. Iteration 102.9 microseconds. Task Construction: Task Tasking CPU Time: 1005.9 microseconds. Task Construction: Task Construction Count: Task Construction: Task Construction Task construction: approximation Task constructi	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Text Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75290.0000 0.0000 Text Iteration 64 75298.0000 75290.0000 0.0000 Text Iteration 64 75298.0000 75290.0000 0.0000
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Literration 1024 73839.0000 73939.0000 Test Literration 1024 Test Literration 1024 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Mail Time: 1005.9 microseconds With one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 1 73949.0000 0.0000 Test Tecration 1 1 73950.0000 0.0000 Test Tecration 2 1 73950.0000 0.0000 Test Tecration 4 1 73950.0000 0.0000 Test Tecration 4 1 73950.0000	75296.0000 75296.0000 0.0000 Text teration 2 75295.0000 75296.0000 0.0000 Teration 4 75297.0000 75297.0000 0.0000 Teration 8 75297.0000 75297.0000 0.0000 Teration 8 75297.0000 75297.0000 0.0000 Teration 16 75298.0000 75298.0000 0.0000 Teration 32 75298.0000 75298.0000 0.0000 Teration 32 75298.0000 75298.0000 0.0000 Teration 32 75298.0000 75298.0000 0.0000 Teration 64 75298.0000 75298.0000 0.0000 Teration 64 75298.0000 75298.0000 0.0000 Teration 64 75298.0000 75298.0000 0.0000 Teration 64 75298.0000 75298.0000 0.0000 Teration 64 75300.0000 75300.0000 0.0000 Teration 128 75300.0000 75300.0000 0.0000
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 Test Name: C000001 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Test Description: 105.9 microseconds Test Oracia and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73930.0000 73949.0000 0.0000 Test Terration 2 73950.0000 73950.0000 0.0000 Test Terration 2 73950.0000 73950.0000 0.0000 Test Terration 4 73950.0000 Test Terration 4 73950.0000 Test Terration 4 73950.0000 Terration 4 73950.0000 1erration 4 73950.0000 73950.0000 0.0000 Terration 4 73950.0000 73950.0000 0.0000 Terration 4 73951.0000 73951.0000 0.0000	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 4 75300.0000 75298.0000 0.0000 Iteration 64 75300.0000 75301.0000 1.0000 Iteration 128 75300.0000 75302.0000 1.0000 Iteration 128 75300.0000 75302.0000 1.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 73839.0000 73947.0000 Test Kame: C000001 CPU Time: 1005.9 microseconds Wall Time: 10000 Test Decordption: 1 Test Decordption: 1 Test Decordption: 1 73950.0000 0.0000 Test Decord Decomo 1 73950.0000 1.0000 Test Decomo 1 73951.0000 0.0000 <td>75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75299.0000 0.0000 Iteration 42 75299.0000 75299.0000 0.0000 Iteration 18 75299.0000 75299.0000 0.0000 Iteration 18 75300.0000 75300.0000 Iteration 28 75300.0000 75300.0000 Iteration 28 Iteration 28 Itera</td>	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75299.0000 0.0000 Iteration 42 75299.0000 75299.0000 0.0000 Iteration 18 75299.0000 75299.0000 0.0000 Iteration 18 75300.0000 75300.0000 Iteration 28 75300.0000 75300.0000 Iteration 28 Iteration 28 Itera
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 Test Iterration 1024 Test Name: C00001 CPU Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 13939.0000 73939.0000 0.0000 Test Iteration 2 1 73939.0000 0.0000 Test terration 2 2 73939.0000 0.0000 Terration 4 2 73939.0000 0.0000 Terration 4 2 73939.0000 1.0000 Terration 4 7 73939.0000 0.0000 Terration 4 7 73939.0000 0.0000 Terration 5 7	75296.0000 75296.0000 0.0000 Test Iteration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 42 75300.0000 75299.0000 0.0000 Iteration 64 75300.0000 75300.0000 1.0000 Iteration 128 75301.0000 75302.0000 1.0000 Iteration 28 75302.0000 75303.0000 1.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 73839.0000 73947.0000 Test Tearation 1024 Test Name: C000001 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds With one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 1 73950.0000 0.0000 Test Tearation 2 7 73950.0000 0.0000 Test Tearation 2 7 73950.0000 0.0000 Test Tearation 4 1.0000 73950.0000 0.0000 Test Tearation 4 1.0000 73951.0000 0.0000 Test Tearation 8 1.0000 73951.0000 0.0000	75296.0000 75296.0000 0.0000 Text Icaration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 22 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75300.0000 75309.0000 0.0000 Iteration 54 75300.0000 75300.0000 Iteration 54 75300.0000 75300.0000 Iteration 54 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 753000 75000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000 Iteration 25 75300.0000 75300.0000
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 108.0000 Test Kames C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 739350.0000 0.0000 Test Teration 1 739350.0000 0.0000 Test Teration 2 739350.0000 0.0000 Test Teration 2 73950.0000 0.0000 Test Teration 2 73950.0000 0.0000 Test Teration 2 73950.0000 0.0000 Test Teration 4 73950.0000 0.0000 Test Teration 4 73951.0000 0.0000 Test Teration 4 73951.0000 0.0000 Test Teration 8 73951.0000 0.0000 Test Teration 8 73951.0000	75296.0000 75296.0000 0.0000 Text icration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 42 75298.0000 75298.0000 0.0000 Iteration 42 75300.0000 75300.0000 0.0000 Iteration 44 75300.0000 75300.0000 0.0000 Iteration 128 75300.0000 75300.0000 1.0000 Test Iteration 128 75301.0000 75302.0000 1.0000 Iteration 256 75302.0000 75305.0000 1.0000 Iteration 256 75304.0000 75305.0000 1.0000
73736.0000 73834.0000 1ceration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Name: C000001 CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Wall Time: 1005.9 microseconds Mail Time: 1005.000 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 0.0000 Test Teration 0 0.0000 73950.0000 0.0000 Test Teration 2 0.0000 Tasto:0000 73950.0000 0.0000 Test Teration 4 73950.0000 0.0000 Test Teration 4 73950.0000 0.0000 Tasto:000 73951.0000 <td>75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Text Iteration 4 75297.0000 75297.000 0.0000 Text Iteration 4 75297.000 75297.000 0.0000 Text Iteration 5 75297.000 75297.000 0.0000 Text Iteration 16 75298.0000 75298.0000 0.0000 Text Iteration 16 75298.0000 75298.0000 0.0000 Text Iteration 32 75298.0000 75298.0000 0.0000 Text Iteration 32 75298.0000 75299.0000 0.0000 Text Iteration 44 75298.0000 75299.0000 0.0000 Text Iteration 44 75298.0000 75299.0000 0.0000 Text Iteration 54 75298.0000 75299.0000 0.0000 Text Iteration 128 75300.0000 75301.0000 1.0000 Text Iteration 128 75300.0007 75303.0000 1.0000 Text Iteration 256 75304.0007 75305.0000 Text Iteration 256 75304.0007 75305.0000</td>	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Text Iteration 4 75297.0000 75297.000 0.0000 Text Iteration 4 75297.000 75297.000 0.0000 Text Iteration 5 75297.000 75297.000 0.0000 Text Iteration 16 75298.0000 75298.0000 0.0000 Text Iteration 16 75298.0000 75298.0000 0.0000 Text Iteration 32 75298.0000 75298.0000 0.0000 Text Iteration 32 75298.0000 75299.0000 0.0000 Text Iteration 44 75298.0000 75299.0000 0.0000 Text Iteration 44 75298.0000 75299.0000 0.0000 Text Iteration 54 75298.0000 75299.0000 0.0000 Text Iteration 128 75300.0000 75301.0000 1.0000 Text Iteration 128 75300.0007 75303.0000 1.0000 Text Iteration 256 75304.0007 75305.0000 Text Iteration 256 75304.0007 75305.0000
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 Test Kamen C000001 CPU Timen 1005.9 Mail Timen 1005.9 Mail Timen 1005.9 Mail Timen 1005.9 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 0.0000 Test terration 1 73939.0000 0.0000 Test terration 2 1 73939.0000 0.0000 Test terration 2 1 73930.0000 0.0000 Teration 4 1 73930.0000 1.0000 Teration 4 1 73930.0000 1.0000 Teration 4 1 73930.0000 1.0000 Teration 4 1 73930.0000 0.0000 Test teration 8 1	75296.0000 75296.0000 0.0000 Text icration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Text icration 8 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 42 75298.0000 75298.0000 0.0000 Iteration 44 75300.0000 75300.0000 0.0000 Iteration 26 75300.0000 75300.0000 0.0000 Iteration 128 75300.0000 75300.0000 1.0000 Iteration 26 75300.0000 75300.0000 1.0000 Iteration 28 75300.0000 75300.0000 1.0000 Iteration 5500.0000 1.0000 Iteration 5500.0000 1.0000 Iteration 5500.0000 1.0000 Iteration 5500 3.0000 Iteration 5500 3.00000 Iteration 5500 3.0000 Iteration 5500 3.0000 Iteration 5500 3.00000 Iteration 5500 3.00000 Iteration 5500 3.00000 Iteration 5500 3.00000 Iteration 5500 3.00000 Iteration 5500 3.00000 Iteration 5500 3.00000 Iter
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Terration 1024 73839.0000 73947.0000 Test Terration 1024 CPU Time 1005.9 microseconds Mall Time 1005.9 microseconds Test Certification Class Mame: Tasking CPU Time 1005.9 microseconds Test Description: Test Certification Test Description: a package, no select statement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Teration 2 1 73950.0000 73950.0000 0.0000 Test Teration 2 2 2 73950.0000 0.0000 1.0000 Test Teration 2 2 2 73950.0000 1.0000 1.0000 Test Teration 2 0.0000 1.0000 Tasso 1.0000 1.0000 1.0000 Tasso 1.0000 1.0000 1.0000 Tasso 1.0000 1.0000 1.00	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Text Iteration 4 75297.0000 75297.0000 0.0000 75297.0000 75297.0000 0.0000 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Text Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75299.0000 0.0000 Iteration 32 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75290.0000 Iteration 54 75308.0007 75301.0000 Iteration 256 75308.0007 75301.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Test Iteration 256 75308.0007 7531.0000 Iteration 512 75308.0007 7531.0000 Iteration 512 75308.0000 75311.0000 Iteration 512 Tiso8.0007 75311.0000 Iteration 512 Tiso8.0000 75311.0000 Iteration 512 Tiso8.0007 75311.0000 Iteration 512 Iteration 512
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73939.0000 Test Kames C000001 CPU Times 1005.9 Piet Test Times 1005.9 Mail Times 1005.9 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 0.0000 Test terration 1 73939.0000 0.0000 Test terration 1 1 73939.0000 0.0000 Test terration 2 1 73939.0000 0.0000 Test terration 2 1 73930.0000 0.0000 Test terration 4 1 73931.0000 1.0000 Test terration 4 1 73931.0000 0.0000 Test terration 5 0.0000 Test terration 16 1.0000 73931.0000 1.0000	75296.0000 75296.0000 0.0000 Text icration 2 75295.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75208.0000 75298.0000 0.0000 Iteration 64 75208.0000 75298.0000 0.0000 Iteration 64 75208.0000 75298.0000 0.0000 Iteration 64 75208.0000 75308.0000 75300.0000 7500.0000 Iteration 256 75304.0000 75308.0000 Iteration 257 75308.0000 75308.0000 Iteration 257 75308.0000 75308.0000 Iteration 257 75308.0000 75308.0000 Iteration 254 75308.0000 75308.0000 Iteration 254 75308.0000 7531.0000 Iteration 254 75308.0000 75308.0000 Iteration 254 Iteration 254 It
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 Test Iterration 1024 Test Name: C00001 CPU Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 13949.0000 73950.0000 13950.0000 Iteration 2 1 73950.0000 0.0000 Teration 4 1 73950.0000 0.0000 Teration 4 1 73950.0000 0.0000 Teration 4 1 73950.0000 0.0000 Teration 8 7 73951.0000 0.0000 Teration 8 7 73951.0000 0.0000 <td>75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75298.0000 Iteration 28 75308.0007 75398.0000 Iteration 128 75308.0007 75308.0000 Iteration 512 75308.0007 75311.0000 Text Iteration 512 75308.0000 75311.0000 Text Iteration 512 75308.0000 75311.0000 Text Iteration 1024 75308.0000 75311.0000 Text Iteration 128 75308.0000 75311.0000 Text Iteration 128 75308.0000 Text Iteration 124 75308.0000 Text Iteration 128 75308.0000 Text Iteration 128 Text Iteration 128 Te</td>	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75299.0000 0.0000 Iteration 64 75298.0000 75298.0000 Iteration 28 75308.0007 75398.0000 Iteration 128 75308.0007 75308.0000 Iteration 512 75308.0007 75311.0000 Text Iteration 512 75308.0000 75311.0000 Text Iteration 512 75308.0000 75311.0000 Text Iteration 1024 75308.0000 75311.0000 Text Iteration 128 75308.0000 75311.0000 Text Iteration 128 75308.0000 Text Iteration 124 75308.0000 Text Iteration 128 75308.0000 Text Iteration 128 Text Iteration 128 Te
73726.0000 73832.0000 54.0000 Iteration 1024 73839.0000 73839.0000 5.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Kame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test teration 1 73949.0000 0.0000 Test teration 2 73350.0000 0.0000 1.0000 Test teration 4 2 73350.0000 1.0000 Test teration 4 1 73950.0000 1.0000 Test teration 4 1 73950.0000 1.0000 Test teration 4 1 73950.000 1.0000 Test teration 8 0.0000 1.0000 73951.0000 1.0000 Test teration 8 0.0000 1.0000 73951.0000 1.0000 Test teration 16 0.0000 73951.00	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75297.0000 0.0000 Terat Iteration 4 75297.0000 75297.0000 0.0000 Terat Iteration 8 75297.0000 75297.0000 0.0000 Terat Iteration 16 75298.0000 75298.0000 0.0000 Terat Iteration 6 75298.0000 75298.0000 0.0000 Terat Iteration 16 75300.0000 75399.0000 0.0000 Terat Iteration 18 75300.0000 75399.0000 0.0000 Terat Iteration 18 75300.0000 75398.0000 0.0000 Terat Iteration 28 75300.0000 75398.0000 1.0000 Terat Iteration 28 75300.0000 75300.0000 1.0000 Teration 275 75300.0000 75300.0000 1.0000 Teration 28 75300.0000 75300.0000 1.0000 Teration 28 75300.0000 75300.0000 Teration 28 75300.0000 75300.0000 1.0000 Teration 28 75300.0000 75300.0000 Teration 28 75300.0000 75300.0000 Teration 28 75300.0000 75300.0000 Teration 28 75300.0000 75300.0000 Teration 28 75300.0000 75300.0000 Teration 28 75300.0000 75300.0000 Teration 28 7530000 75300.0000 Teration 28 7530000 75300.0000 Teration 28 7530000 7530000 Teration 28 75300000 75300.0000 Teration 22 7530000 75300.0000 Teration 22 7530000 7530000 Teration 22 7530000 7530000 Teration 22 7530000 7530000 Teration 12 7530000 7530000 Teration 12 7530000 7530000 Teration 12 7530000 7530000 7530000 7530000 Teration 12 7530000 75300000 Teration 12 7530000 7530000 Teration 12 75300000 7530000 Teration 12 753000000000000000000000000000000000000
73780.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 108.0000 Test Terration 1024 Test 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 739350.0000 0.0000 Test Creation 10 739350.0000 739350.0000 0.0000 Teration 2 739350.0000 739350.0000 0.0000 Teration 4 739351.0000 739351.0000 0.0000 Teration 8 739351.0000 739351.0000 0.0000 Teration 16 73953.0000 739351.0000 0.0000 Teration 16 73953.0000 739351.0000 0.0000 Teration 16 73953.0000 73953.0000 0.0	75296.0000 75296.0000 0.0000 Text Teration 2 0.0000 Teration 4 0.0000 Teration 4 0.0000 Teration 75297.0000 0.0000 Teration 75297.0000 0.0000 Teration 8 0.0000 Teration 75297.0000 0.0000 Teration 8 0.0000 Teration 8 0.0000 Teration 9 0.0000 Teration 16 0.0000 Teration 16 0.0000 Teration 16 0.0000 Teration 16 0.0000 Teration 17 0.0000 Totar 0000 75298.0000 0.0000 Teration 32 75298.0000 Totar 0000 75299.0000 0.0000 Teration 64 75299.0000 Totar 10000 75300.0000 0.0000 Teration 2789.0000 0.0000 Teration 28 0.0000 Totar 10000 1.0000 Teration 256 1.0000 Table 20000 1.0000 Teration 512 75308.0000 Table 20000 3.0000 Teration 512 75308.0000 Table 20000 3.0000 Teration 512 75308.0000 Table 20000 5.0000 Te
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Kame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73930.0000 0.0000 Test Teartion 10 0.0000 Test Teartion 2 73930.0000 0.0000 1.0000 Test Teartion 2 73950.0000 0.0000 Test Teartion 2 73950.0000 1.0000 73950.0000 0.0000 Test Teartion 4 73951.0000 73950.0000 0.0000 Test Teartion 4 73951.0000 73950.0000 0.0000 Test Teartion 8 73951.0000 73951.0000 1.0000 Test Teartion 8 73951.0000 73951.0000 1.0000 Test Teartion 16 73953.0000 73955.0000 1.00000 Test Teartion 16	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Iteration 4 75297.0000 0.0000 Text Icration 8 0.0000 0.0000 Text Icration 15297.0000 0.0000 0.0000 Total 1cration 15298.000 0.0000 0.0000 Total 1cration 32 0.0000 0.0000 Teration 54 0.0000 0.0000 Total 0.0000 1.0000 1.0000 Teration 256 7303.0000 1.0000 Total 0.0000 1.0000 1.0000 Terat Icration 512 1.0000
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 Test Kames C000001 CPU Times 1005.9 microseconds Mall Times 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 73939.0000 73930.0000 0.0000 Test Tecration 2 733950.0000 73930.0000 73950.0000 0.0000 Teration 4 73951.0000 73931.0000 73951.0000 0.0000 Test Teration 8 73951.0000 739351.0000 0.0000 Teration 16 73953.0000 73953.0000 1.0000 Teration 16 73953.0000	75296.0000 75296.0000 0.0000 Text Icration 2 75295.0000 0.0000 Iteration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 8 75297.0000 0.0000 Text Icration 16 75298.0000 0.0000 Text Icration 32 75298.0000 0.0000 Text Icration 32 75298.0000 0.0000 Text Icration 64 75300.0000 0.0000 Text Icration 128 75300.0000 0.0000 Text Icration 236 75300.0000 0.0000 Text Icration 24 1.0000 1.0000 Text Icration 256 75300.0000 3.0000 Text Icration 512 75300.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 73839.0000 73947.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Test 0 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, 10000 Iteration 1 73949.0000 0.0000 Test Test 100 1 0000 Test 1000 73950.0000 0.0000 Test 1000 73950.0000 0.0000 Testion 2 7350.0000 0.0000 Testion 4 73351.0000 0.0000 Testion 5 1.0000 1.0000 Testion 6 73951.0000 1.0000 Testion 16 73951.0000 1.0000 Testion 16 73953.0000 1.0000 Testion 16 73955.0000	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Iteration 4 75297.0000 0.0000 Text Icration 7 0.0000 Text Icration 8 0.0000 Text Icration 9 0.0000 Text Icration 10 0.0000 Text Icration 12 0.0000 Total 1000 0.0000 Text Icration 12 0.0000 Total 1000 0.0000 Text Icration 12 0.0000 Total 1000 0.0000 Text Icration 12 0.0000 Total 000 0.0000 Text Icration 12 0.0000 Total 0000 0.0000 Text Icration 22 0.0000 Total 0000 0.0000 Text Icration 128 1.0000 Total 0000 1.0000 Text Icration 256 1.0000 Total 0000 1.0000 Text Icration 512 1.0000 Total 0000 1.0000 Total 0000 1.0000 Total 0000 1.0000 Total 0000 1
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 108.0000 Test Kamen C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Test create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Test terration 1 73939.0000 73930.0000 0.0000 Test terration 2 73939.0000 73950.0000 0.0000 Test terration 2 73939.0000 73950.0000 0.0000 Test terration 2 73950.0000 73950.0000 0.0000 Test terration 4 73951.0000 73951.0000 0.0000 Test terration 8 73951.0000 73951.0000 0.0000 Test terration 1 73951.0000 73951.0000 0.0000 Test terration 1 73951.0000 73951.0000 0.0000 Test terration 1 73951.0000 73951.0000 0.0000 Test terration 16 <td>75296.0000 75296.0000 0.0000 Text Icration 2 75295.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 8 75297.0000 0.0000 Text Icration 8 75297.0000 0.0000 Text Icration 16 75297.0000 0.0000 Text Icration 16 75298.0000 0.0000 Text Icration 16 75298.0000 0.0000 Text Icration 32 75298.0000 0.0000 Text Icration 54 75300.0000 0.0000 Text Icration 128 1.0000 1.0000 Text Icration 256 3.0000 1.0000 Text Icration 256 3.0000 1.0000 Text Icration 128 3.0000 1.0000 Text Icreration 124 5.0000 <t< td=""></t<></td>	75296.0000 75296.0000 0.0000 Text Icration 2 75295.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 8 75297.0000 0.0000 Text Icration 8 75297.0000 0.0000 Text Icration 16 75297.0000 0.0000 Text Icration 16 75298.0000 0.0000 Text Icration 16 75298.0000 0.0000 Text Icration 32 75298.0000 0.0000 Text Icration 54 75300.0000 0.0000 Text Icration 128 1.0000 1.0000 Text Icration 256 3.0000 1.0000 Text Icration 256 3.0000 1.0000 Text Icration 128 3.0000 1.0000 Text Icreration 124 5.0000 <t< td=""></t<>
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Name: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Maintrian Wall Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entrise, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Teration 1 73949.0000 0.0000 Test Teration 2 10000 Test time: 10000 0.0000 Testion 2 10000 10000 Testion 2 1.0000 1.0000 Testion 4 73951.0000 0.0000 Testion 4 73951.0000 0.0000 Testion 5 1.0000 1.0000 Testion 5 1.0000 1.0000 Testion 6 73951.0000 1.0000 Testion 16 73951.0000 1.0000 Testion 16 73951.0000 1.0000 Testion	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Iteration 4 75297.0000 0.0000 Text Icration 7 0.0000 Text Icration 8 0.0000 Text Icration 9 0.0000 Text Icration 1297.0000 0.0000 Text Icration 1297.0000 0.0000 Text Icration 1297.0000 0.0000 Text Icration 16 0.0000 Text Icration 1297.0000 0.0000 Text Icration 1297.0000 0.0000 Text Icration 1297.0000 0.0000 Text Icration 120000 0.0000 Text Icration 12 0.0000 To 200.0000 0.0000 Text Icration 128 0.0000 To 300.0000 1.0000 Text Icration 128 1.0000 To 300.0000 1.0000 Text Icration 256 1.0000 To 300.0000 1.0000 Text Icration 124 7.0000 To 300.0000 1.0000 Text Icration 512 7.0000 To 300000075305.0000 1.0000
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 108.0000 Test Kamen C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 0.0000 Test treation 1 73939.0000 0.0000 Test treation 1 73939.0000 0.0000 Test treation 2 73930.0000 1.0000 Test treation 4 73931.0000 1.0000 Test treation 8 73951.0000 1.0000 Test treation 6 73951.0000 1.0000 Test treation 8 73951.0000	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75296.0000 0.0000 Iteration 4 75297.0000 75297.0000 0.0000 Iteration 8 75297.0000 75297.0000 0.0000 Iteration 16 75297.0000 75297.0000 0.0000 Iteration 16 75298.0000 75298.0000 0.0000 Iteration 32 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 64 75298.0000 75298.0000 0.0000 Iteration 128 75298.0000 75298.0000 0.0000 Iteration 128 75298.0000 75298.0000 0.0000 Iteration 128 75300.0000 75298.0000 0.0000 Iteration 128 75300.0000 75300.0000 0.0000 Iteration 128 75300.0000 75300.0000 1.0000 Iteration 255 75304.0000 75305.0000 1.0000 Iteration 255 75304.0000 75305.0000 1.0000 Iteration 255 75304.0000 75305.0000 1.0000 Iteration 254 75305.0000 75305.0000 Iteration 254 75306.0000 7531.0000 Iteration 254 75306.0000 75316.0000 Iteration 2048 75333.0000 Iteration 2048 75333.0000 75345.0000 Iteration 2048 75333.0000 7545.0000 Iteration 2048 75333.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Name: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Maintrian Wall Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement Iteration 1024 with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Iteration 2 0.0000 1.0000 73950.0000 1.0000 1.0000 173950.0000 0.0000 1.0000 Teration 4 73351.0000 0.0000 Test Iteration 8 0.0000 Test Iteration 16 0.0000 Test Iteration 16 1.0000 Test Iteration 16 73951.0000 Test Iteration 16 73951.0000 Test Iteration 16 73951.0000 Test Iteration 16 73951.0000 <td< td=""><td>75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 4 0.0000 75297.0000 Total 1 cration 4 0.0000 75297.0000 Total 1 cration 4 0.0000 75297.0000 Total 1 cration 4 0.0000 Terrat 1 con 75297.0000 0.0000 Terration 16 0.0000 Terration 16 0.0000 Terration 16 0.0000 Total 1 control 75297.0000 0.0000 Terration 16 0.0000 Total 1 control 75297.0000 0.0000 Terration 16 0.0000 Total 0000 75299.0000 0.0000 Terration 24 75300.0000 Total 0000 75301.0000 1.0000 Terration 256 75000 Total 0000 75303.0000 1.0000 Terration 512 75000 Total 0000 75305.0000 1.0000 Terration 512 75000 Total 0000 75305.0000 1.0000 <td< td=""></td<></td></td<>	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 4 0.0000 75297.0000 Total 1 cration 4 0.0000 75297.0000 Total 1 cration 4 0.0000 75297.0000 Total 1 cration 4 0.0000 Terrat 1 con 75297.0000 0.0000 Terration 16 0.0000 Terration 16 0.0000 Terration 16 0.0000 Total 1 control 75297.0000 0.0000 Terration 16 0.0000 Total 1 control 75297.0000 0.0000 Terration 16 0.0000 Total 0000 75299.0000 0.0000 Terration 24 75300.0000 Total 0000 75301.0000 1.0000 Terration 256 75000 Total 0000 75303.0000 1.0000 Terration 512 75000 Total 0000 75305.0000 1.0000 Terration 512 75000 Total 0000 75305.0000 1.0000 <td< td=""></td<>
73736.0000 54.0000 Territion 1024 73834.0000 73839.0000 Test Literation 1024 73839.0000 73939.0000 Test Kame: C000001 CPU Time: 1005.9 Pict Test Conseconds Iteration Wall Time: 1005.9 Test Conseconds Iteration Count: With one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 0.0000 Test treation 1 73949.0000 0.0000 Test treation 2 739350.0000 0.0000 Test treation 2 739350.0000 0.0000 Test treation 2 739350.0000 0.0000 Test treation 4 73950.0000 73950.0000 0.0000 Testion 4 73950.0000 73950.0000 0.0000 Testion 4 73951.0000 73951.0000 0.0000 Testiceration 16 <t< td=""><td>75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75297.0000 0.0000 Terat Iteration 4 75297.0000 75297.0000 0.0000 Terat Iteration 8 75297.0000 75297.0000 0.0000 Terat Iteration 16 75298.0000 75298.0000 0.0000 Terat Iteration 22 75298.0000 75298.0000 0.0000 Terat Iteration 32 75299.0000 75298.0000 0.0000 Terat Iteration 4 75300.0000 75398.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 1.0000 Terat Iteration 52 75309.0007 5298.0000 1.0000 Terat Iteration 52 75309.0007 5298.0000 1.0000 Terat Iteration 52 75309.0007 5398.0000 1.0000 Terat Iteration 52 75309.0007 5310.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 Terat Iteration 52 Terat Iteration 52 Terat I</td></t<>	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75297.0000 0.0000 Terat Iteration 4 75297.0000 75297.0000 0.0000 Terat Iteration 8 75297.0000 75297.0000 0.0000 Terat Iteration 16 75298.0000 75298.0000 0.0000 Terat Iteration 22 75298.0000 75298.0000 0.0000 Terat Iteration 32 75299.0000 75298.0000 0.0000 Terat Iteration 4 75300.0000 75398.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 0.0000 Terat Iteration 52 75309.0000 75298.0000 1.0000 Terat Iteration 52 75309.0007 5298.0000 1.0000 Terat Iteration 52 75309.0007 5298.0000 1.0000 Terat Iteration 52 75309.0007 5398.0000 1.0000 Terat Iteration 52 75309.0007 5310.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 75309.0007 75300.0000 Terat Iteration 52 Terat Iteration 52 Terat Iteration 52 Terat I
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 Test Name: C000001 CPU Time: 1005.9 microseconds Mall Time: 1005.000 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 0.0000 Task Creation 10 0.0000 73950.0000 0.0000 Teration 4 1.0000 Task Creation 4 73950.0000 Task Coolo 73951.0000 0.0000 Teration 8 73951.0000 Task Coolo 73951.0000 0.0000 Test Teration 16	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 4 0.0000 Totation 4 0.0000 Text Icration 4 0.0000 Totat Icration 5 0.0000 Terration 16 0.0000 Totas 0000 75298.0000 0.0000 Terration 16 0.0000 Totas 0000 75298.0000 0.0000 Terration 64 75290.0000 Totas 0000 75299.0000 0.0000 Terration 128 75301.0000 Totas 0000 75301.0000 1.0000 Terration 226 75301.0000 Totas 0000 75303.0000 1.0000 Terration 226 75301.0000 Totas 0000 75303.0000 1.0000 Terration 226 75301.0000 Totas 0000 75303.0000 1.00000 Totas 0000 75303.0
73730.0000 54.0000 Terrtion 1024 73834.0000 73839.0000 Test Iterration 1024 73839.0000 73947.0000 108.0000 Test Kame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test teration 1 73949.0000 0.0000 Test teration 1 73950.0000 0.0000 Test teration 2 73950.0000 0.0000 Test teration 2 73950.0000 0.0000 Test teration 2 73950.0000 0.0000 Test teration 4 73951.0000 1.0000 Test teration 1 0.0000 73951.0000 1.0000 Test teration 1 0.0000 73951.0000 1.0000 Test teration 2 0.0000 73952.0000 1.0000 Tes	75296.0000 75296.0000 0.0000 Textica 2 75296.0000 75297.0000 0.0000 Textica 4 75297.0000 75297.0000 0.0000 Textica 4 75297.0000 75297.0000 0.0000 Textica 4 75297.0000 75297.0000 0.0000 Textica 8 75297.0000 75297.0000 0.0000 Textica 8 0.0000 Textica 16 0.0000 Textica 128 0.0000 Textica 128 0.0000 Textica 128 0.0000 Textica 128 1.0000 Textica 128 1.0000 Textica 128 1.0000 Textica </td
73730.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Tierration 1024 73839.0000 73947.0000 108.0000 Test Tierration 1025.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 739350.0000 0.0000 Test Creation 1 739350.0000 739350.0000 0.0000 Teration 2 73950.0000 73950.0000 0.0000 Teration 4 73951.0000 73951.0000 1.0000 Teration 8 73951.0000 73951.0000 0.0000 Teration 16 73951.0000 73951.0000 1.0000 Teration 16 73951.0000 73951.0000 0.0000 Teraticration 16 73955.0000	75296.0000 75296.0000 0.0000 Text Icration 2 0.0000 75296.0000 75297.0000 0.0000 Text Icration 4 0.0000 Text Icration 4 0.0000 Text Icration 4 0.0000 Totation 4 0.0000 Text Icration 4 0.0000 Totation 16 0.0000 Totation 16 0.0000 Text Iteration 16 0.0000 Totation 12 1.0000 Totation 12 1.0000 Totation 12 1.0000
¹⁷ 3796.0007 3795.0000 54.0000 Terration 1024 73833.0000 73947.0000 108.0000 Test Kame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Mail Time: 1005.9 microseconds. Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 73949.0000 0.0000 Test Teration 2 73950.0000 73950.0000 1.0000 Test Teration 2 73950.0000 73950.0000 0.0000 Test Teration 4 73951.0000 73950.0000 0.0000 Test Teration 7 73951.0000 73950.0000 0.0000 Test Teration 8 73951.0000 73950.0000 Test Teration 8 73951.0000 73950.0000 Test Teration 8 73951.0000 73950.0000 Test Teration 8 73950.0000 73950.0000 Test Teration 8 73950.0000 73950.0000 Test Teration 16 73950.0000 73950.0000 Test Teration 12 73950.0000 73950.0000 Test Teration 22 73955.0000 73950.0000 Test Teration 23 73950.0000 73950.0000 Test Teration 24 73950.0000 73950.0000 Test Teration 25 73950.0000 73950.0000 Test Teration 26 73950.0000 73950.0000 Test Teration 28 73950.0000 73950.0000 Test Teration 28	75296.0000 75296.0000 0.0000 Text Licration 2 75296.0000 0.0000 Text Licration 4 75297.0000 0.0000 Text Licration 8 0.0000 10000 Text Licration 15297.0000 0.0000 10000 Text Licration 15297.0000 0.0000 10000 Text Licration 15297.0000 0.0000 10000 Text Licration 32 0.0000 10000 Text Licration 54 0.0000 10000 Text Licration 23 1.0000 1.0000 Text Licration 24 1.0000 1.0000 Text Licration 256 1.0000 1.0000 Text Licration 512 1.0000 1.0000 Text Licration 1024 1.0000 1.0000
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Tierration 1024 73839.0000 73947.0000 108.0000 Test Tierration 1025 microseconds Mall Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Test create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Test terration 1 73939.0000 73949.0000 0.0000 Test terration 2 73939.0000 73950.0000 1.0000 Terration 2 73939.0000 73950.0000 0.0000 Test terration 4 73939.0000 73950.0000 0.0000 Test terration 4 73939.0000 73951.0000 0.0000 Test terration 8 73939.0000 73951.0000 0.0000 Test terration 16 73939.0000 73951.0000 1.0000 Test terration 16 73939.0000 73951.0000 0.0000 Test terration 16 <	75296.0000 75296.0000 0.0000 Text Teration 2 0.0000 Tister teration 4 0.0000 Text Teration 5 0.0000 Total 7.000 75297.0000 0.0000 Text Teration 8 0.0000 Text Teration 16 0.0000 Text Teration 16 0.0000 Text Teration 16 0.0000 Text Teration 18 0.0000 Total 0000 75299.0000 0.0000 Teration 12 0.0000 Total 0000 75299.0000 0.0000 Teration 12 0.0000 Total 0000 75301.0000 1.0000 Test Teration 12 1.0000 Total 0000 75301.0000 1.0000 Test Teration 255 1.0000 Tet Teration 124 1.0000 Tet Teration 124 1.0000 Tet Teration 254 1.0000 Tet Teration 124 1.0000 Total 0000 753
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Kame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73930.0000 0.0000 Test Teration 2 73930.0000 0.0000 Iteration 2 73930.0000 0.0000 Test Teration 2 73930.0000 0.0000 Iteration 4 73930.0000 0.0000 Test Teration 4 73950.0000 0.0000 Iteration 4 73951.0000 1.0000 Test Teration 8 73951.0000 1.0000 Test Teration 8 73951.0000 1.0000 Test Teration 8 73951.0000 1.0000 Test Teration 16 73951.0000 1.0000 Test Teration 12 73955.0000 0.0000	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75297.0000 0.0000 Text Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Text Iteration 8 75298.0000 75298.000 0.0000 Text Iteration 32 75298.0000 75298.000 0.0000 Text Iteration 32 75298.0000 75299.000 0.0000 Text Iteration 64 75298.0000 75299.000 0.0000 Text Iteration 64 75298.0000 75299.000 0.0000 Text Iteration 54 75298.0000 75299.000 0.0000 Text Iteration 54 75298.0000 75299.000 0.0000 Text Iteration 54 75298.0000 75299.0000 0.0000 Text Iteration 54 75300.0000 75301.0000 1.0000 Text Iteration 256 75301.0000 75303.0000 Text Iteration 256 75304.0007 75305.0000 Text Iteration 256 75304.0007 75305.0000 Text Iteration 264 75305.0000 75305.0000 Text Iteration 274 75305.0000 75305.0000 Text Iteration 264 75304.0007 75305.0000 Text Iteration 274 75305.0000 75305.0000 Text Iteration 274 75305.0000 75305.0000 Text Iteration 274 75305.0000 75475.0000 Text Iteration 204 75305.0000 75475.0000 Text Iteration 204 75305.0000 75475.0000 Text Iteration 204 75305.0000 75475.0000 Text Iteration 204 75305.0000 75475.0000 Text Iteration 264 75305.0000 75475.0000 Text Iteration 264 75305.0000 7557.0000 Text Iteration 1584 75351.0000 7557.0000 Text Iteration 1584 75551.0000 7557.0000 5.0000
73726.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Tierration 1024 73839.0000 73947.0000 108.0000 Test Tierration 1025 microseconds Mall Time: 1005.9 microseconds. Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Terration 1 73949.0000 0.0000 Test Terration 1 739350.0000 1.0000 Terration 2 73950.0000 73950.0000 Terration 2 73950.0000 0.0000 0.0000 Terration 4 73950.0000 73950.0000 1.0000 Terration 4 73950.0000 1.0000 Terration 4 73951.0000 73950.0000 Terration 1 73950.0000 1.0000 1.0000 Terration 8 73951.0000 Terration 16 73951.0000 1.0000 Terration 16 73951.0000 73951.0000 Terration 17 73951.0000	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 75297.0000 0.0000 Text Iteration 4 75297.0000 75297.0000 0.0000 Text Iteration 8 75297.0000 75297.000 0.0000 Text Iteration 8 75297.0000 75297.0000 0.0000 Text Iteration 16 75298.0000 75298.000 0.0000 Text Iteration 32 75298.0000 75299.000 0.0000 Text Iteration 32 75298.0000 75299.0000 0.0000 Text Iteration 44 75298.0000 75299.0000 0.0000 Text Iteration 52 75298.0000 75299.0000 0.0000 Text Iteration 44 75298.0000 75299.0000 0.0000 Text Iteration 54 75298.0000 75299.0000 0.0000 Text Iteration 54 75298.0000 75299.0000 0.0000 Text Iteration 54 75298.0000 75399.0000 1.0000 Text Iteration 54 75304.0000 75301.0000 1.0000 Text Iteration 128 75304.0000 75301.0000 1.0000 Text Iteration 512 75304.0000 75311.0000 3.0000 Text Iteration 512 75305.0000 75333.0000 1.0000 Text Iteration 255 75311.0000 7531.0000 5.0000 Text Iteration 254 75303.0000 75333.0000 1.0000 Text Iteration 254 75304.0000 75333.0000 1.0000 Text Iteration 1024 75303.0000 753345.0000 1.0000 Text Iteration 204 75333.0000 75345.0000 12.0000 Text Iteration 204 75333.0000 75345.0000 12.0000 Text Iteration 204 75333.0000 75345.0000 12.0000 Text Iteration 1024 75333.0000 75345.0000 12.0000 Text Iteration 4096 75334.0000 75354.0000 24.0000 Text Iteration 4096 75335.0000 75354.0000 Text Iteration 4096 75345.0000 75354.0000 24.0000 Text Iteration 4096 75345.0000 75354.0000 Text Iteration 4096 75345.0000 75354.0000 Text Iteration 16384 75349.0000 75354.0000 Text Iteration 16384 75459.0000 75354.0000 Text Iteration 16384 75459.0000 75354.0000 Text Iteration 16384 75459.0000 75451.0000 Text I
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 5.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Name: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Testion 2 73950.0000 0.0000 Iteration 2 73950.0000 Test Testion 2 0.0000 1.0000 Test Testion 4 0.0000 73950.0000 0.0000 Testion 4 1.0000 73950.000 0.0000 Testion 5 73951.0000 0.0000 Testion 7 73951.0000 0.0000 Testion 7 73951.0000 73951.0000 0.0000 Testion 7 73951.0000 Testion 16 1.0000 Testion 7 73951.0000 Testion 16 73951.0000 1.0000 <tr< td=""><td>75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75297.0000 0.0000 Iteration 75297.0000 0.0000 Text Iteration 75297.0000 0.0000 Terat Iteration 75297.0000 0.0000 Terat Iteration 10 75297.0000 75298.0000 0.0000 Iteration 22 75298.0000 75298.000 0.0000 Iteration 32 75298.0000 75299.000 0.0000 Iteration 32 75298.0000 75299.000 0.0000 Iteration 64 75298.0000 75299.000 0.0000 Iteration 64 75298.0000 75299.000 0.0000 Iteration 128 75300.0007 75299.000 0.0000 Iteration 25 75300.0007 75299.000 0.0000 Iteration 28 75300.0007 75300.0000 1.0000 Iteration 128 75300.0007 75300.0000 Iteration 256 75300.0007 75300.0000 1.0000 Iteration 256 75300.0007 75310.0000 1.0000 Iteration 124 7531.0000 75310.0000 1.0000 Iteration 1024 7531.0000 75310.0000 1.0000 Iteration 1024 7531.0000 7531.0000 1.0000 Iteration 1024 7531.0000 7531.0000 1.0000 Iteration 1024 7531.0000 7531.0000 1.0000 Iteration 1024 7533.0000 75334.0000 1.0000 Iteration 1024 7533.0000 7534.0000 1.0000 Iteration 1024 7533.0000 7534.0000 1.0000 Iteration 1024 7533.0000 7534.0000 1.0000 Iteration 1024 7533.0000 75334.0000 1.0000 Iteration 1024 7533.0000 753.0000 24.0000 Iteration 1024 7533.0000 753.0000 5557.0000 5557.0000 Iteration 1534 75551.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 5500</td></tr<>	75296.0000 75296.0000 0.0000 Text Iteration 2 75296.0000 75297.0000 0.0000 Iteration 75297.0000 0.0000 Text Iteration 75297.0000 0.0000 Terat Iteration 75297.0000 0.0000 Terat Iteration 10 75297.0000 75298.0000 0.0000 Iteration 22 75298.0000 75298.000 0.0000 Iteration 32 75298.0000 75299.000 0.0000 Iteration 32 75298.0000 75299.000 0.0000 Iteration 64 75298.0000 75299.000 0.0000 Iteration 64 75298.0000 75299.000 0.0000 Iteration 128 75300.0007 75299.000 0.0000 Iteration 25 75300.0007 75299.000 0.0000 Iteration 28 75300.0007 75300.0000 1.0000 Iteration 128 75300.0007 75300.0000 Iteration 256 75300.0007 75300.0000 1.0000 Iteration 256 75300.0007 75310.0000 1.0000 Iteration 124 7531.0000 75310.0000 1.0000 Iteration 1024 7531.0000 75310.0000 1.0000 Iteration 1024 7531.0000 7531.0000 1.0000 Iteration 1024 7531.0000 7531.0000 1.0000 Iteration 1024 7531.0000 7531.0000 1.0000 Iteration 1024 7533.0000 75334.0000 1.0000 Iteration 1024 7533.0000 7534.0000 1.0000 Iteration 1024 7533.0000 7534.0000 1.0000 Iteration 1024 7533.0000 7534.0000 1.0000 Iteration 1024 7533.0000 75334.0000 1.0000 Iteration 1024 7533.0000 753.0000 24.0000 Iteration 1024 7533.0000 753.0000 5557.0000 5557.0000 Iteration 1534 75551.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 550.0000 Iteration 1534 75561.0000 7557.0000 5500
73736.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Tierration 1024 73839.0000 73939.0000 Test Tierration 1024 CPU Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 73950.0000 0.0000 Test Tieration 2 73950.0000 73950.0000 0.0000 Teration 4 73950.0000 73950.0000 0.0000 Test tieration 8 73951.0000 73951.0000 0.0000 Testion 12 0.0000 <	75296.0000 75296.0000 0.0000 Text Licration 2 75295.0000 0.0000 Tertion 4 75297.0000 0.0000 Text Licration 4 75297.0000 0.0000 Text Licration 8 75297.0000 0.0000 Text Licration 8 75297.0000 0.0000 Text Text Lion 8 0.0000 1.0000 Text Text Lion 16 0.0000 1.0000 Text Text Lion 16 0.0000 1.0000 Text Text Lion 16 0.0000 1.0000 Text Text Lion 12 0.0000 1.0000 Text Lieration 12 0.0000 1.0000 Text Lieration 12 0.0000 1.0000 Text Lieration 22 1.0000 1.0000 Text Lieration 28 1.0000 1.0000 Toolo 0.000 7503.0000 1.0000 1.0000 Text Lieration 224 1.0000 1.0000 Toolo 0.000 7503.0000 1.0000 1.0000 Text Lieration 1024 1.0000 1.0000 Text Lieration 1024 1.0000 1.0000 Toolo 0.000 75033.0000 1.0000 1.0000 </td
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Tearation 1024 73839.0000 73947.0000 108.0000 Test Tearation 1005.9 microseconds Iteration Count: 1024 Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Teaction 1 73949.0000 0.0000 12000 Test Teaction 1 73950.0000 0.0000 12000 Teration 4 73950.0000 0.0000 12000 Test Tearation 4 1.0000 10000 12000 Teration 5 1.0000 10000 12000 Test Tearation 16 1.0000 13351.0000 1.0000 Test Tearation 16 1.0000 13351.0000 1.0000 Test Tearation 16 13353.0000 1.0000 13355.000 1.0000 T	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 7 0.0000 75297.0000 Text Icration 16 0.0000 Text Icration 16 0.0000 Text Icration 16 0.0000 Text Icration 32 75298.0000 75297.0000 75299.0000 0.0000 Teration 128 75290.0000 75000 75299.0000 0.0000 Teration 28 75300.0000 75000 75297.0000 0.0000 Teration 28 75300.0000 75000 75299.0000 0.0000 Teration 28 75300.0000 75000 75297.0000 1.0000 Teration 28 1.0000 75000 7529.0000 1.0000 Teration 296 1.0000
73726.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Kame: C000001 Class Name: Tasking CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 1.0000 Test Creation 1 73949.0000 0.0000 Test teration 1 73950.0000 0.0000 Test teration 2 73950.0000 0.0000 Test teration 2 73950.0000 1.0000 Test teration 4 73951.0000 1.0000 Test teration 4 73951.0000 1.0000 Test teration 1 0.0000 73952.0000 1.0000 Test teration 6 73953.0000 1.0000 73952.0000 73952.0000 1.0000 Test teration 1 0.0000	75296.0000 75296.0000 0.0000 Text Licration 2 75296.0000 0.0000 Text Licration 4 75297.0000 0.0000 Text Licration 8 75297.0000 0.0000 Text Licration 16 0.0000 10000 Text Licration 16 0.0000 10000 Text Licration 20 0.0000 10000 Text Licration 30 0.0000 10000 Text Licration 4 75297.0000 0.0000 Text Licration 22 0.0000 0.0000 Text Licration 32 10000 0.0000 Text Licration 54 0.0000 1.0000 Totol 0000 7501.0000 1.0000 1.0000 Test Licration 226 1.0000 1.0000 Totol 0000 7503.0000 1.0000 1.0000 Text Licration 1024 1.0000 1.0000 Totol 0000 7503.0000 1.0000 1.0000 Text Licration 1024 1.0000 1.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 5.0000 Test Iteration 1024 73839.0000 73947.0000 108.0000 Test Name: 000059 microseconds Mall Time: 1005.9 microseconds Mall Time: 10000 noncecture using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Tearation 10 0.0000 73950.0000 73950.0000 0.0000 Teration 4 73950.0000 0.0000 Test Tearation 8 1.0000 Test Tearation 16 1.0000 Test Tearation 16 73950.0000 Test Tearation 16 73950.0000 Test Tearation 16 739	75296.0000 75296.0000 0.0000 Text Icration 2 75296.0000 0.0000 Terration 4 75297.0000 0.0000 Text Icration 4 75297.0000 0.0000 Text Icration 5 0.0000 75297.0000 Total Icration 5 0.0000 1 Total Icration 16 0.0000 1 Total Icration 22 0.0000 1 Total Icration 32 7.0000 0.0000 Total Icration 32 0.0000 1.0000 Total Icration 32 0.0000 1.0000 Total Icration 1024 1.0000
73736.0000 54.0000 Iteration 1024 73834.0000 73839.0000 Test Tieration 1024 73839.0000 73947.0000 108.0000 Test Tearation 1024 CPU Time: 1005.9 microseconds Iteration Count: 1024 Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73949.0000 0.0000 Test Tearation 73939.0000 73950.0000 0.0000 Test Tearation 2 73350.0000 73939.0000 73950.0000 0.0000 Test Tearation 4 73950.0000 73950.0000 73951.0000 0.0000 Test Tearation 8 73950.0000 73951.0000 1.0000 Test Tearation 8 0.0000 Test Tearation 8 73951.0000 1.0000 Test Tearation 8 73953.0000 Test Test 10 73951.0000 1.0000 Test Test 10 1.0000 Test Test 10 73951.0000 1.0000 Test Tearation 8 <t< td=""><td>75296.0000 75296.0000 0.0000 Text Licration 2 75296.0000 0.0000 Text Licration 4 75297.0000 0.0000 Text Licration 8 75297.0000 0.0000 Text Licration 16 0.0000 10000 Text Licration 16 0.0000 10000 Text Licration 20 0.0000 10000 Text Licration 30 0.0000 10000 Text Licration 4 75297.0000 0.0000 Text Licration 20 0.0000 10000 Text Licration 32 75398.0000 0.0000 Text Licration 54 0.0000 1.0000 Toolo 0000 75301.0000 1.0000 1.0000 Text Licration 226 1.0000 1.0000 Toolo 0000 75303.0000 1.0000 1.0000 Text Licration 1024 1.0000 1.0000 Toolo 0000 75303.0000 1.0000 1.0000 Text Licration 1024 1.0000 1.0000</td></t<>	75296.0000 75296.0000 0.0000 Text Licration 2 75296.0000 0.0000 Text Licration 4 75297.0000 0.0000 Text Licration 8 75297.0000 0.0000 Text Licration 16 0.0000 10000 Text Licration 16 0.0000 10000 Text Licration 20 0.0000 10000 Text Licration 30 0.0000 10000 Text Licration 4 75297.0000 0.0000 Text Licration 20 0.0000 10000 Text Licration 32 75398.0000 0.0000 Text Licration 54 0.0000 1.0000 Toolo 0000 75301.0000 1.0000 1.0000 Text Licration 226 1.0000 1.0000 Toolo 0000 75303.0000 1.0000 1.0000 Text Licration 1024 1.0000 1.0000 Toolo 0000 75303.0000 1.0000 1.0000 Text Licration 1024 1.0000 1.0000
73726.0000 54.0000 Terration 1024 73834.0000 73839.0000 Test Tierration 1024 73839.0000 73947.0000 108.0000 Test Tierration 1005.9 microseconds Mall Time: 1005.9 microseconds Mall Time: 1005.9 microseconds Test Create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Iteration 1 73939.0000 0.0000 Test Teration 2 73950.0000 0.0000 Teration 2 73950.0000 0.0000 Teration 4 73951.0000 1.0000 Teration 4 73951.0000 0.0000 Teration 6 73951.0000 0.0000 Teration 16 73951.0000 1.0000 Teration 16 73951.0000 73951.0000 73951.0000 73951.0000 1.0000 Teration 16 73951.0000 73950.0000 73955.0000 73950.0000 1.0000 </td <td>75296.0000 75296.0000 0.0000 Teatration 2 75296.0000 0.0000 Teatration 4 75297.0000 0.0000 Teatration 4 75297.0000 0.0000 Teatration 5 0.0000 75297.000 Totat Teartion 4 0.0000 Totat Teartion 5 0.0000 Totat Teartion 16 0.0000 Teat Teartion 28 0.0000 Totas 0000 75299.0000 0.0000 Teat Teartion 28 0.0000 Totas 0000 75299.0000 0.0000 Teat Teartion 28 1.0000 Totas 0000 75301.0000 1.0000 Teat Teartion 28 1.0000 Totas 0000 75303.0000 1.0000 Teat Teartion 28 1.0000 Totas 0000 75303.0000 1.0000 Teat Teartion 24 1.0000 Totas 0000 75303.0000 1.0000 </td>	75296.0000 75296.0000 0.0000 Teatration 2 75296.0000 0.0000 Teatration 4 75297.0000 0.0000 Teatration 4 75297.0000 0.0000 Teatration 5 0.0000 75297.000 Totat Teartion 4 0.0000 Totat Teartion 5 0.0000 Totat Teartion 16 0.0000 Teat Teartion 28 0.0000 Totas 0000 75299.0000 0.0000 Teat Teartion 28 0.0000 Totas 0000 75299.0000 0.0000 Teat Teartion 28 1.0000 Totas 0000 75301.0000 1.0000 Teat Teartion 28 1.0000 Totas 0000 75303.0000 1.0000 Teat Teartion 28 1.0000 Totas 0000 75303.0000 1.0000 Teat Teartion 24 1.0000 Totas 0000 75303.0000 1.0000

task defined and used in procedure, no select statement, no loop

Iteration 1 75659.0000 75659.0000 0.0000

Terration 75141.0000 0.0000 76142.0000 76142.0000 0.0000 76142.0000 76143.0000 1.0000 Terration 16 76143.0000 76144.0000 1.0000 Terration 32 76143.0000 76144.0000 0.0000 Terration 32 76145.0000 76146.0000 1.0000 Terration 64 76145.0000 76174.0000 1.0000 Terration 128 76145.0000 76174.0000 1.0000 Terration 256 76150.0000 76175.0000 1.0000 Terration 256 76150.0000 76175.0000 1.0000 Terration 256 76160.0000 76175.0000 2.0000 Terration 1024 76175.0000 76374.0000 104.0000 Terration 264 76270.0000 76374.0000 104.0000 Terration 454.1 micromeconde Mail Time, 454.1 micromeconde Mail Time 454.0000 0.0000 Terration 2 Terration 2	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Test Iteration 4 77940.0000 77940.0000 Test Iteration 8 77941.0000 77941.0000 Test Iteration 8 77941.0000 77942.0000 Test Iteration 16 77942.0000 77942.0000 Test Iteration 16 77944.0000 77944.0000 Test Iteration 22 77948.0000 77948.0000 Test Iteration 25 77970.0000 77972.0000 Test Iteration 25 77962.0000 7895.0000 Test Iteration 1024 70002.0000 7805.0000 Test Iteration 1024 70002.0000 7805.0000 Test Iteration 25 77962.0000 7895.0000 Test Iteration 25 77962.0000 7895.0000 Test Iteration 512 77962.0000 7895.0000 Test Iteration 512 77962.0000 7895.0000 Test Iteration 512 77962.0000 Fiber 1024 70000 Test Iteration 512 70000 Test Iteration 512 70000 Test Iteration 512 70952.0000 Fiber 1024 70000 Test Iteration 1024 70052.0000 Fiber 1024 Fiber 1024 Fiber 1024 Fiber 1024 Fiber 1024 Fiber 1024 Fiber
Terration 76142.0000 0.0000 76142.0000 76142.0000 0.0000 7614 (1654) 76143.0000 76143.0000 1.0000 Terration 16 76143.0000 76144.0000 1.0000 Terration 32 76143.0000 76144.0000 1.0000 Terration 32 76143.0000 76145.0000 1.0000 Terration 64 76145.0000 76174.0000 1.0000 Terration 128 76145.0000 76151.0000 1.0000 Terration 256 76150.0000 76151.0000 1.0000 Terration 256 76150.0000 76153.0000 1.0000 Terration 256 76150.0000 76153.0000 1.0000 Terration 256 76150.0000 76153.0000 1.0000 Terration 256 76150.000 76153.0000 1.0000 Terration 256 76150.000 76173.0000 1.0000 Terration 312 76173.000 76173.0000 2.0000 Terration 312 76173.000 76175.0000 2.0000 Terration 2677.0000 5.0000 Terration 270.0000 Terration 270.0000 Terration 270.0000 Terration 270.0000 Terration 270.0000 Terration 270.0000 Terration 270.0000 Terration 270.0000 Terration 2045 76270.0000 76374.0000 104.0000 Terration 244 76270.0000 76374.0000 104.0000 Terration 245 76270.0000 75376.0000 104.0000 Terration 11 Terration 1 Terration	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Test Iteration 4 77940.0000 77940.0000 Test Iteration 8 77941.0000 77941.0000 Test Iteration 8 77941.0000 77942.0000 Test Iteration 16 77942.0000 77942.0000 Test Iteration 32 77944.0000 77948.0000 Test Iteration 32 77945.0000 77948.0000 Test Iteration 4 77955.0000 0.0000 Test Iteration 54 77955.0000 13.0000 Test Iteration 28 77955.0000 13.0000 Test Iteration 28 77955.0000 7395.0000 Test Iteration 28 77955.0000 7395.0000 Test Iteration 26 77955.0000 7395.0000 Test Iteration 128 77955.0000 7395.0000 Test Iteration 256 77970.0000 7395.0000 Test Iteration 256 77970.0000 7395.0000 Test Iteration 512 77955.0000 5402.0000 Test Iteration 512 77955.0000 5402.0000 Test Iteration 512 77955.0000 540.0000 Test Iteration 1024 70000.0000 Test Iteration 512 77955.0000 540.0000 Test Iteration 512 77955.0000 55.0000 Test Iteration 512 77955.0000 7595.0000 Test Iteration 512 77955.0000 Test Iteration 1024 78052.0000 7802.0000 Test Iteration 0024 78052.0000 7802.0000 Test Iteration 0024 78052.0000 7802.0000 Test Iteration 1024 78052.0000 7802.0000 Test Iteration 0024 Test Iteration 0024 Test Iteration 0024 Test Iteration 0024 Test Iteration 00
Testicology 714.0000 0.0000 Testicology 76142.0000 0.0000 Testicology 76142.0000 1.0000 Iteration 16 16.0000 1.0000 Testicology 76143.0000 0.0000 Testicology 76144.0000 1.0000 Testicology 76144.0000 0.0000 Testicology 76144.0000 1.0000 Testicology 76144.0000 1.0000 Testicology 76145.0000 1.0000 Testicology 76145.0000 1.0000 Testicology 76151.0000 76150.0000 1.0000 Testicology 76151.0000 76150.0000 1.0000 Testicology 76150.0000 2.0000 Testicology 76150.0000 2.0000 Testicology 76150.0000 1.0000 Testicology 76150.0000 5.0000 Testicology 76150.0000 Testicology 76150.0000 Testicology 76150.0000 Testicology 76150.0000 Testicology 76150.0000 Testicology 76170.0000 Testicology 7600 Testicology	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Test Iteration 4 77940.0000 77940.0000 Test Iteration 8 77941.0000 77941.0000 Test Iteration 8 77941.0000 77942.0000 Test Iteration 64 77942.0000 77942.0000 Test Iteration 64 77943.0000 77948.0000 Test Iteration 64 77945.0000 77948.0000 Test Iteration 64 77945.0000 77948.0000 Test Iteration 52 77955.0000 130.0000 Test Iteration 28 7795.0000 77948.0000 Test Iteration 52 77955.0000 130.0000 Test Iteration 52 77955.0000 130.0000 Test Iteration 52 77959.0000 20000 Test Iteration 52 77959.0000 7595.0000 Test Iteration 52 77959.0000 20000 Test Iteration 52 77959.0000 20000 Test Iteration 51 77959.0000 540.0000 Test Iteration 51 77959.0000 7595.0000 Test Iteration 51 77950.0000 7595.0000 Test Iteration 50 Test Iteration 51 77950.0000 7595.0000 Test Iteration 51 77950.0000 75950.0000 Test Iteration 50 Test Iteration 51 77950.0000 75950.0000 Test Iteration 51 77950.0000 759500 Test Iteration 51 7795000 Test Iteration 51 7795000 7795000 7795000 7795000 7795000 7795000 7795000 7
Testain 1000 7111.0000 0.0000 76142.0000 76143.0000 0.0000 76142.0000 76143.0000 1.0000 Iteration 16 0.0000 Test Iteration 16 0.0000 Test Iteration 32 0.0000 Test Iteration 32 0.0000 Test Iteration 32 0.0000 Test Iteration 32 0.0000 Iteration 64 0.0000 Test Iteration 64 0.0000 Iteration 64 0.0000 Test Iteration 128 0.0000 Iteration 255 0.0000 7.5000 Iteration 1024 0.0000 Test Iteration 2048 76270.0000 75174.0000 104.0000 Test Mame: T000001 Class Mame: Teal Mainimm rendervous, entry call end return time I task I entry, task inside procedure no select	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77938.0000 77939.0000 Test Iteration 8 77940.0000 77940.0000 Test Iteration 8 77941.0000 77942.0000 Iteration 32 77942.0000 77942.0000 Iteration 32 77943.0000 77944.0000 Iteration 32 77944.0000 77948.0000 Iteration 32 77945.0000 70948.0000 Iteration 32 77955.0000 1.0000 Iteration 52 77955.0000 1.0000 Iteration 52 77959.0000 2.0000 Iteration 512 77959.0000 3.0000 Iteration 512 77959.0000 540.0000 Iteration 512 77950.0000 54.0000 Iteration 512 77950.0000 54.0000 Iteration 512 77950.0000 54.0000 Iteration 1024 78050.0000 108.0000 Iteration 512 7956.0000 54.0000 Iteration 512 7956.1 microseconds Iteration Contr Test Iteration 1024 78050.0000 Test. 1 microseconds Iteration Contr Test Control Test Iteration 1024 78050.0000 Test.0000 Iteration 50 Iteration 512 7956.1 microseconds Iteration Contr Test Control Test Iteration 1024 78050.0000 Test.0000 7800.000 Test Iteration 1024 78050.0000 Test.0000 54.0000 Test Iteration 256 79970.0000 Test.0000 54.0000 Test Iteration 256 79970.0000 Test.0000 54.0000 Test Iteration 1024 78050.0000 Test.0000 Test.0000 Test.0000 Test Iteration 1024 78050.0000 Test.0000 Test.0000 Test Iteration 1024 78050.0000 Test.0000 Test.0000 Test.0000 Test.0000 Test.0000 Test.00000 Test.00000 Test.0000 Test.00000 Test.00000 Test.0000 Test.00000 Test.00000 Test.00000 Test.00000 Test.000000 Test.
Tetration 0 76142.0000 0.0000 76142.0000 76142.0000 0.0000 Tetration 8 15143.0000 76143.0000 1.0000 Tetration 16 76143.0000 76144.0000 1.0000 Tetration 32 76143.0000 76144.0000 0.0000 Tetration 32 76143.0000 76146.0000 1.0000 Tetration 64 76143.0000 76145.0000 1.0000 Tetration 25 76145.0000 76155.0000 1.0000 Tetration 128 76150.0000 76155.0000 1.0000 Tetration 256 76150.0000 76155.0000 1.0000 Tetration 256 76150.0000 76155.0000 1.0000 Tetration 256 76150.0000 76155.0000 1.0000 Tetration 256 76150.000 76155.0000 1.0000 Tetration 256 76160.000 76175.0000 2.0000 Tetration 128 76173.000 76259.0000 1.0000 Tetration 1024 7620.0000 76259.0000 5.0000 Tetration 265 7620.000 76259.0000 5.0000 Tetration 264 7620.0000 76259.0000 104.0000 Tetration 2048 Tetration 2048 Tetration 2048 Tetration 2049 Tetration 2049 Tetration 454.1 microseconds Tetration Count: Tetration Count: Tetration Count: Tetration 2041 microseconds Tetration Count: Tetration 2041 microseconds Tetration Count: Tetration	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77939.0000 77940.0000 Test Iteration 6 77941.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Test Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 12 77944.0000 77944.0000 0.0000 Test Iteration 22 77948.0000 77948.0000 0.0000 Test Iteration 25 77975.0000 77979.0000 13.0000 Test Iteration 25 7797.0000 7797.0000 7399.0000 Test Iteration 1024 77962.0000 7807.0000 54.0000 Test Iteration 1024 78062.0000 78170.0000 108.0000 Test Iteration 1024 78062.0000 78170.0000 108.0000 Test Iteration 1024 78062.0000 7810.0000 108.0000 Test Iteration 1024 78062.0000 7810.0000 108.0000 Test Iteration 1024 78062.0000 7810.0000 108.0000 Test Iteration 1024 78062.0000 7817.0000 108.0000 Test Iteration 1024 78062.0000 7810.0000
Tetration 75141.0000 0.0000 75142.0000 75142.0000 0.0000 75142.0000 75142.0000 1.0000 Iteration 8 75143.0000 75143.0000 1.0000 Test Iteration 32 75143.0000 75144.0000 0.0000 Test Iteration 32 75143.0000 75144.0000 1.0000 Test Iteration 32 75145.0000 75145.0000 1.0000 Test Iteration 64 75145.0000 75151.0000 1.0000 Test Iteration 128 75145.0000 75151.0000 1.0000 Test Iteration 256 75150.0000 75150.0000 1.0000 Test Iteration 256 75150.0000 75150.0000 1.0000 Test Iteration 256 75150.0000 75150.0000 1.0000 Test Iteration 256 75170.000 75152.0000 2.0000 Test Iteration 312 75170.000 75175.0000 2.0000 Test Iteration 1024 75207.0000 75374.0000 110.000 Test Iteration 264 75270.0000 75374.0000 104.0000 Test Iteration 454.1 microseconds Wall Time, 454.1 microseconds Wall Time, 454.1 microseconds	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77939.0000 77940.0000 Test Iteration 8 77941.0000 77941.000 0.0000 Test Iteration 8 77941.0000 77942.000 1.0000 Iteration 16 77942.0000 77942.000 0.0000 Test Iteration 12 77944.0000 77944.0000 0.0000 Test Iteration 22 77948.0000 77948.0000 0.0000 Test Iteration 25 7797.0000 7797.0000 13.0000 Test Iteration 25 7797.0000 7895.0000 2.0000 Test Iteration 12 7798.0000 780.0000 154.0000 Test Iteration 12 7798.0000 780.000 108.0000 Test Iteration 12 7799.000 7895.0000 54.0000 Test Iteration 25 7797.0000 7895.0000 54.0000 Test Iteration 12 77985.0000 78170.0000 Test Iteration 124 78062.0000 78170.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 25 77950.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 1024 78062.0000 Test Iteration 55.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 1024 78062.0000 78170.0000 Test Iteration 1024 78062.0000 Test Iteration 1024 780750.0000 Test Iteration 1024 780750.0000 Test Iteration 1
Tetration 0 10000 0.0000 Tetration 0 10000 0.0000 Total2.0000 76143.0000 0.0000 Tetration 16 1000 0.0000 Tetration 16 1000 0.0000 Tetration 32 0.0000 Tetration 32 0.0000 Tetration 32 0.0000 Tetration 32 0.0000 Tetration 64 10000 Tetration 128 0.0000 Tetration 124 0.0000 Tetration 1024 0.0000 Tetration 2048 76270.0000 75174.0000 Tetration 2048 76270.0000 75174.0000 Class Mase, Teal	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77938.0000 77939.0000 Test Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 6 77944.0000 77944.0000 0.0000 Test Iteration 6 77948.0000 77948.0000 0.0000 Test Iteration 6 77948.0000 77955.0000 0.0000 Test Iteration 28 7795.0000 77955.0000 0.0000 Test Iteration 28 7795.0000 77972.0000 13.0000 Iteration 256 77970.0000 7597.0000 13.0000 Iteration 512 77959.0000 7595.0000 2.0000 Iteration 128 77959.0000 7595.0000 3.0000 Test Iteration 128 77959.0000 7595.0000 3.0000 Test Iteration 128 77959.0000 7595.0000 3.0000 Iteration 128 77959.0000 7595.0000 13.0000 Iteration 128 77959.0000 5602.0000 54.0000 Iteration 1024 76050.0000 54.0000 54.0000 Iteration 1024 76050.0000 108.0000
76141.0000 0.0000 76142.0000 6142.0000 76142.0000 76143.0000 76142.0000 76143.0000 176142.0000 76143.0000 176142.0000 76143.0000 176142.0000 76143.0000 176143.0000 76143.0000 176143.0000 76144.0000 176143.0000 76144.0000 176143.0000 76144.0000 176143.0000 76144.0000 176143.0000 76143.0000 176143.0000 76143.0000 176143.0000 76143.0000 176143.0000 76143.0000 176143.0000 76143.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0000 76140.0000 176143.0	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77938.0000 77939.0000 Test Iteration 6 77940.0000 77940.0000 Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77944.0000 2.0000 Iteration 2 77945.0000 77944.0000 0.0000 Test Iteration 6 77948.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 7555.0000 6.0000 Iteration 128 77955.0000 73955.0000 13.0000 Iteration 256 77970.0000 73972.0000 13.0000 Iteration 512 77959.0000 7595.0000 2.0000 Iteration 128 77959.0000 7395.0000 3.0000 Test Iteration 52 70000 7395.0000 3.0000 Iteration 128 70000 7395.0000 3.0000 Iteration 128 70950.0000 7395.0000 3.0000 Iteration 128 70950.0000 3.0000 Iteration 128 70950.0000 3.0000 Iteration 128 70950.0000 3.0000 Test Iteration 512 70950.0000 54.0000 Iteration 512 70950.0000 54.0000 Iteration 1024 70000 5000 54.0000 Iteration 1024 70500.0000 54.0000 Test Iteration 1024 70500.0000 54.0000
Terration 8 76142.0000 76142.0000 0.0000 76142.0000 76143.0000 1.0000 Terration 16 76143.0000 76143.0000 1.0000 Terration 16 76143.0000 76144.0000 1.0000 Terration 32 76145.0000 76145.0000 1.0000 Terration 64 76146.0000 76150.0000 1.0000 Terration 64 76146.0000 76150.0000 1.0000 Terration 27 76150.0000 76150.0000 1.0000 Terration 25150.0000 7.0000 76150.0000 75173.0000 7.0000 Terration 512 76160.0000 75173.0000 1.0000 Terration 512 76160.0000 75173.0000 2.0000 Terration 52 76160.0000 76173.0000 2.0000 Terration 52 76160.0000 76173.0000 2.0000 Terration 52 76160.0000 76173.0000 2.0000 Terration 52 76175.0000 76173.0000 2.0000 Terration 52 76175.0000 76173.0000 2.0000 Terration 52 76175.0000 76173.0000 2.0000 Terration 52 76175.0000 76173.0000 2.0000 76175.0000 76173.0000 2.0000 76175.0000 7627.0000 5.0000 76175.0000 7627.0000 5.0000 76175.0000 7627.0000 5.0000 76175.0000 7627.0000 1.0000 Terration 52 76175.0000 7627.0000 5.0000 76175.0000 7627.0000 7627.0000 5.0000 76175.0000 7627.0000 7627.0000 5.0000 76175.0000 7627.0	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Test Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 1.0000 Tration 16 77942.0000 77942.0000 0.0000 Test Iteration 32 77944.0000 77944.0000 0.0000 Test Iteration 32 77948.0000 77948.0000 3.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Test Iteration 52 77949.0000 77948.0000 0.0000 Test Iteration 52 77949.0000 77948.0000 0.0000 Test Iteration 52 77949.0000 77948.0000 0.0000 Test Iteration 52 77955.0000 13.0000 755.0000 0.0000 Test Iteration 52 77959.0000 77950.0000 0.0000 Test Iteration 52 77959.0000 77950.0000 2.0000 Test Iteration 52 77959.0000 7599.0000 2.0000 Test Iteration 512 77959.0000 759.0000 3.0000 Test Iteration 512 77959.0000 759.0000 3.0000 Test Iteration 512 77959.0000 759.0000 54.0000 Test Iteration 512 77950.0000 7592.0000 54.0000 Test Iteration 512 77959.0000 7562.0000 54.0000 Test Iteration 512 77952.0000 54.0000 54.0000
Terration 8 76142.0000 76142.0000 0.0000 76142.0000 76143.0000 1.0000 76142.0000 76143.0000 1.0000 76143.0000 76144.0000 1.0000 76145.0000 76144.0000 1.0000 76145.0000 76144.0000 1.0000 76145.0000 76144.0000 1.0000 76145.0000 76147.0000 1.0000 76145.0000 76147.0000 1.0000 Terration 128 76150.0000 76153.0000 1.0000 Terration 256 76150.0000 76153.0000 1.0000 Terration 256 76150.0000 76173.0000 1.0000 Terration 256 76150.0000 76173.0000 1.0000 Terration 512 76150.0000 76173.0000 1.0000 Terration 512 76150.0000 76173.0000 1.0000 Terration 512 76173.000 76173.0000 76173.0000 Terration 1024 76202.0000 76207.0000 5.0000 Terration 1024 76207.0000 76207.0000 5.0000 Terration 2048 7625.0000 7625.0000 1.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77940.0000 0.0000 Iteration 8 77941.0000 77941.000 0.0000 Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77943.0000 77944.0000 2.0000 Iteration 32 77944.0000 77944.0000 0.0000 Iteration 32 77948.0000 77948.0000 0.0000 Iteration 64 77949.0000 77955.0000 0.0000 Iteration 128 77955.0000 77955.0000 0.0000 Iteration 28 77970.0000 77972.0000 13.0000 Iteration 28 77970.0000 77972.0000 2.0000 Iteration 28 77972.0000 77972.0000 2.0000 Iteration 28 77972.0000 77972.0000 2.0000 Iteration 25 77979.0000 77959.0000 2.0000 Iteration 25 77979.0000 77959.0000 3.0000 Iteration 25 77979.0000 77959.0000 3.0000 Iteration 25 77979.0000 77959.0000 3.0000 Iteration 25 77979.0000 77959.0000 3.0000 Iteration 512 78002.0000 78056.0000 54.0000
/8141.0000 7611.0000 0.0000 76142.0000 76142.0000 0.0000 76142.0000 76143.0000 1.0000 16ert 1 cration 9 76143.0000 76143.0000 0.0000 16ert 10 76144.0000 0.0000 16ert 10 76144.0000 1.0000 16ert 10 76144.0000 76140.0000 176145.0000 76146.0000 1.0000 176145.0000 76147.0000 1.0000 176147.0000 76150.0000 1.0000 176147.0000 76153.0000 1.0000 176150.0000 76153.0000 2.0000 176173.000 76173.0000 2.0000 176174.000 76173.0000 76173.0000 176175.0000 76173.0000 76173.0000 176175.0000 76173.0000 2.0000 176175.0000 76173.0000 2.0000 176175.0000 76173.0000 2.0000 176175.0000 76259.0000 2.0000 176175.0000 7600 5.0000 176175.0000 7600 5.0000 176175.	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77940.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 32 77944.0000 77944.0000 2.0000 Iteration 32 77944.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Iteration 64 77949.0000 77955.0000 0.0000 Iteration 128 77955.0000 77955.0000 13.0000 Iteration 28 77970.0000 77972.0000 7399.0000 Iteration 28 77970.0000 77972.0000 7395.0000 Iteration 32 77979.0000 7395.0000 13.0000 Iteration 35 77979.0000 7399.0000 2.0000 Iteration 512
Tetration 5 Tetration 6 Tetration 6 Tetration 6 Tetration 6 Tetration 6 Tetration 6 Tetration 16 Tetration 16 Tetration 16 Tetration 32 Tetration 5 Tetration 5 Tetration 6 Tetration 128 Tetration 128 Tetration 128 Tetration 256 Tetration 5 Tetration 7 Tetration	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 2.0000 Iteration 32 77944.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Test Iteration 128 77940.0000 77955.0000 0.0000 Test Iteration 128 7795.0000 77970.0000 13.0000 Iteration 128 77970.0000 77972.0000 7397.0000 Iteration 256 77972.0000 77972.0000 2.0000
7614.1000 7614.2000 0.0000 7614.2000 7614.2000 0.0000 Territor 16 0.0000 Territor 32 0.0000 Territor 32 0.0000 Territor 32 0.0000 Territor 32 0.0000 Territor 64 0.0000 Territor 64 0.0000 Territor 128 0.0000 <t< td=""><td>77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77940.0000 77940.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77942.0000 77944.0000 2.0000 Iteration 32 77944.0000 77948.0000 0.0000 Test Iteration 32 77948.0000 77948.0000 0.0000 Test Iteration 32 77948.0000 77948.0000 0.0000 Iteration 6 77949.0000 77948.0000 0.0000 Test Iteration 5 77949.0000 77948.0000 0.0000 Test Iteration 32 77949.0000 77948.0000 0.0000 Test Iteration 5 77949.0000 77955.0000 1.0000 Test Iteration 12 77949.0000 77955.0000 1.0000</td></t<>	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77940.0000 77940.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77942.0000 77944.0000 2.0000 Iteration 32 77944.0000 77948.0000 0.0000 Test Iteration 32 77948.0000 77948.0000 0.0000 Test Iteration 32 77948.0000 77948.0000 0.0000 Iteration 6 77949.0000 77948.0000 0.0000 Test Iteration 5 77949.0000 77948.0000 0.0000 Test Iteration 32 77949.0000 77948.0000 0.0000 Test Iteration 5 77949.0000 77955.0000 1.0000 Test Iteration 12 77949.0000 77955.0000 1.0000
18:14:1000 10:1000 18:14:1000 10:1000 18:14:1000 10:1000 18:14:1000 10:000 18:14:1000 10:000 18:14:1000 10:000 18:14:1000 10:000 19:11:1000 10:000 19:11:1000 10:0	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77938.0000 77939.0000 Test Iteration 4 77940.0000 77940.0000 Iteration 8 77941.0000 77941.0000 Test Iteration 2 77942.0000 77942.0000 Iteration 16 77942.0000 77942.0000 Iteration 2 77944.0000 77944.0000 Test Iteration 6 77948.0000 77948.0000 Test Iteration 6 77955.0000 7.055.0000 Iteration 128 7795.0000 7.055.0000 Iteration 128 7795.0000 7.0000 Test Iteration 2 7795.0000 7.0000 Test Iteration 2 7795.0000 7.0000 Test Iteration 2 7795.0000 1.0000 Test Iteration 2 7795.0000 2.0000
76141.0000 0.0000 76142.0000 76142.0000 76142.0000 76142.0000 76142.0000 76143.0000 76142.0000 76143.0000 76142.0000 76143.0000 76143.0000 0.0000 Peet Iteration 16 76143.0000 76144.0000 76143.0000 0.0000 Peet Iteration 32 76145.0000 76146.0000 1.0000 76146.0000 Teration 64 76145.0000 76146.0000 1.0000 76146.0000 Teration 64 76150.0000 7.0000 Teration 1.0000 Teration 64 76150.0000 3.0000 Teration 1.0000 Teration 1.0000 Teration 1.0000 Teration 28 76150.0000 7.0000 Teration 28 76151.0000 7.0000 Teration 256 76160.0000 7513.0000 12.0000	77939.0000 77939.0000 Test Iteration 2 77939.0000 77939.0000 Iteration 4 77939.0000 77939.0000 Test Iteration 4 77940.0000 77940.0000 Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 1.0000 Test Iteration 16 77942.0000 77944.0000 0.0000 Test Iteration 2 77943.0000 77944.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 7555.0000 6.0000 Iteration 128 77955.0000 7300.0000
76141.0000 0.0000 76142.0000 76142.0000 76142.0000 76143.0000 76142.0000 76143.0000 76142.0000 1.0000 Terration 16 76143.0000 0.0000 Peet Iterration 16 76143.0000 1.0000 Terration 32 76145.0000 76144.0000 Peet Iterration 32 76145.0000 76145.0000 Terration 64 76145.0000 76145.0000 Terration 64 76145.0000 76145.0000 Terration 64 76150.0000 761000 Terration 64 76150.0000 761000 Terration 64 76150.0000 76150.0000 Terration 1.0000 Terration 28 76150.0000 7150.0000 Terration 25 76150.0000 7199.0000 Terration 25 76150.0000 7199.0000 Terration <td>77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77941.0000 77940.0000 0.0000 Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 2 77945.0000 77944.0000 0.0000 Iteration 3 77945.0000 77948.0000 0.0000 Iteration 64 77945.0000 77948.0000 0.0000 Iteration 28 77945.0000 7755.0000 6.0000</td>	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77941.0000 77940.0000 0.0000 Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 2 77945.0000 77944.0000 0.0000 Iteration 3 77945.0000 77948.0000 0.0000 Iteration 64 77945.0000 77948.0000 0.0000 Iteration 28 77945.0000 7755.0000 6.0000
78141.0000 0.0000 76142.0000 76142.0000 76142.0000 76142.0000 76142.0000 76143.0000 76142.0000 1.0000 Tearation 1 76142.0000 1.0000 Tearation 1 76143.0000 0.0000 Tearation 1 76144.0000 1.0000 Tearation 2 76145.0000 76144.0000 76146.0000 1.0000 Tearation 3 76146.0000 76146.0000 76146.0000 76146.0000 76146.0000 76150.0000 76151.0000 76151.0000 76151.0000 76150.0000 76151.0000 76150.0000 76151.0000 76150.0000 76151.0000 76150.0000 76151.0000 76150.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Iteration 8 77941.0000 77941.0000 0.0000 Iteration 8 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Iteration 16 77942.0000 77944.0000 2.0000 Iteration 32 77944.0000 77948.0000 3.0000 Iteration 32 77944.0000 77948.0000 0.0000 Test Iteration 64 77949.0000 77948.0000 0.0000 Iteration 64 77949.0000 77948.0000 0.0000 Iteration 64
/siti.0000 /511.0000 0.0000 /6122.0000 /5142.0000 0.0000 Peet Iteration 9 76142.0000 /5143.0000 1.0000 Peet Iteration 16 76143.0000 /5144.0000 1.0000 Peet Iteration 26 76145.0000 /5144.0000 1.0000 76147.0000 /5147.0000 1.0000 Feet Iteration 64 76147.0000 /5150.0000 1.0000 Teration 128 76150.0000 /5151.0000 1.0000 Teration 128 76150.0000 /5154.0000 /5154.0000 Teration 128 76150.0000 /5154.0000 /5154.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Iteration 8 77941.0000 77941.0000 0.0000 Iteration 8 77942.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Iteration 16 77942.0000 77944.0000 2.0000 Iteration 32 77945.0000 77948.0000 3.0000 Iteration 32 77944.0000 77948.0000 0.0000 Iteration 6 77944.0000 77948.0000 0.0000 Iteration 54
/stations / / / / / / / / / / / / / / / / / / /	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77940.0000 0.0000 Iteration 8 77941.0000 77941.0000 0.0000 Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 2.0000 Iteration 32 77944.0000 7.0000 1.0000 Iteration 32 77944.0000 7.0000 1.0000 Iteration 32 77944.0000 7.0000 1.0000 Iteration 32 77944.0000 7.0000 1.0000
101411000 0.0000 101411000 0.0000 101411000 1.0000 101421000 76143.0000 101421000 76143.0000 101421000 76143.0000 101421000 76143.0000 1014310000 76143.0000 1014310000 76143.0000 1014310000 76144.0000 1014310000 76144.0000 1014310000 76144.0000 1014310000 76143.0000 1014310000 76143.0000 1014310000 76143.0000 1014310000 76143.0000 1014310000 76143.0000 1014010000 76143.0000 1014010000 76143.0000 1014010000 76143.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77940.0000 0.0000 Iteration 8 77940.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77944.0000 77944.0000 2.0000 Iteration 32 77944.0000 7.0000 Test Iteration 32 77944.0000 7.0000
/5141.0000 /5141.0000 0.0000 76142.0000 76142.0000 0.0000 rest Iteration 8 76142.0000 76143.0000 1.0000 rest Iteration 16 76143.0000 76144.0000 1.0000 rest Iteration 32 76144.0000 76144.0000 0.0000 rest Iteration 32 76146.0000 76144.0000 1.0000 rest Iteration 64 76146.0000 76141.0000 1.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77938.0000 77939.0000 0.0000 Test Iteration 8 77940.0000 77940.0000 0.0000 Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 0.0000 Test Iteration 16 77942.0000 77942.0000 0.0000
76141.0000 76142.0000 0.0000 76142.0000 76142.0000 0.0000 Teart Tteration 8 76142.0000 76143.0000 1.0000 Teartion 16 76143.0000 76144.0000 0.0000 Teartion 32 76144.0000 76144.0000 0.0000 Teartion 32 76145.0000 76146.0000 1.0000 Teartion 64	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 8 77940.0000 77940.0000 0.0000 Test Iteration 8 77941.0000 77942.0000 1.0000 Iteration 16 77942.0000 77942.0000 0.0000 Iteration 16 77942.0000 77944.0000 2.0000
76141.0000 76142.0000 0.0000 76142.0000 76142.0000 0.0000 Tear I turation 8 76142.0000 76143.0000 1.0000 Teartion 16 76143.0000 76144.0000 0.0000 Teartion 32 76144.0000 76144.0000 0.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77930.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 17942.0000 1.0000 Iteration 16 77942.0000 0.7942.0000 0.0000
/siti.0000 /siti.0000 0.0000 76142.0000 76142.0000 0.0000 Fest Tieration 8 76142.0000 76143.0000 1.0000 Terration 76143.0000 0.0000 Fest Tieration 16 76143.0000 76144.0000 1.0000 Teeration 32	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Iteration 8 77941.0000 77941.0000 0.0000 Test Iteration 8 77941.0000 17942.0000 1.0000
/5141.0000 /5141.0000 0.0000 76142.0000 /5142.0000 0.0000 76142.0000 /5143.0000 1.0000 76143.0000 /5143.0000 0.0000 Feet learation 16 76143.0000 /5143.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Iteration 8 77941.0000 7941.0000 0.0000
76142.0000 76143.0000 0.0000 Test Iteration 8 Test Iteration 8 76142.0000 76143.0000 1.0000 Iteration 16 76143.0000 76143.0000 0.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 4 77940.0000 77940.0000 0.0000 Iteration 8
76142.0000 76142.0000 0.0000 76142.0000 76142.0000 0.0000 Fest Iteration 8 76142.0000 76143.0000 1.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4 77939.0000 77939.0000 0.0000 Test Iteration 4
lteration 8 76142.0000 76142.0000 0.0000	77939.0000 77939.0000 0.0000 Test Iteration 2 77939.0000 77939.0000 0.0000 Iteration 4
78141.0000 /8141.0000 0.0000	77939.0000 77939.0000 0.0000 Test Iteration 2
76141 0000 76141 0000 0 0000	77939 0000 77939 0000 0 0000
76141.0000 76141.0000 0.0000	Iteration 2
75141.0000 76141.0000 0.0000 Exertion 4	Test Iteration 1 77938.0000 77939.0000 1.0000
76140.0000 76141.0000 1.0000 Feat Iteration 2	Iteration 1 77938.0000 77938.0000 0.0000
Iteration 2	Thomas 1
Test Iteration 1 76140,0000 76140,0000 0.0000	Average Whetstone rating : 353 XWIPS
Iteration 1 76140.0000 76140.0000 0.0000	Average time per cycle : 2834.67 milliseconds
74	
10 parameters, in INTEGER	
Procedure call and return time measurement	Reinhold P. Weicker's DHRYSTOWE composite benchmark
Wall Time: 23.2 microseconds. Iteration Count: Test Description:	16384 1.2000 is time in milliseconds for one Dhrystone Test Description:
CPU Time: 23.2 microseconds	Test Name: A000091 Class Name: Composite
Cost Mamos	
Test Iteration 16384 76001.0000 76138.0000 137.0000	Clock Resolution (average) = 1.00000000000000 seconds Clock Resolution (variance) = 0.00000000000000 seconds
75902.0000 76001.0000 99.0000	Funct of sample values is 7000 Clock Resolution = 1.0000000000000 seconds
75833.0000 75902.0000 69.0000	Number of semile values is 7000
75784.0000 75833.0000 49.0000	Determine clock resolution using second differences of values returned by the function CPU Time Clock.
75749.0000 75783.0000 34.0000 Iteration 8192	Clock resolution measurement running Test Description:
Test Iteration 4096	Test Name: A000090
Iteration 4096	
Test Iteration 2048 75707.0000 75724.0000 17.0000	One tasks active, two entries, tasks in a package using select statement
75694.0000 75707.0000 13.0000	Task entry call and return time measured
75686.0000 75694.0000 8.0000	Wall Time: 566.4 microseconds. Iteration Count:
75679.0000 75685.0000 6.0000	Test Name: T000004 Class Name: Taskin CPU Time: 566.4 microseconds
75675.0000 75679.0000 4.0000 Eteration 1024	76513.0000 76634.0000 121.0000
Pest Iteration 512	Test Iteration 1024
Iteration 512 75671.0000 75675.0000 4.0000	Iteration 1024 76507.0000 76512.0000 5.0000
Test Iteration 256 75669.0000 75671.0000 2.0000	Test Iteration 512 76446.0000 76507.0000 61.0000
75667.0000 75669.0000 2.0000	76443.0000 76446.0000 3.0000 Text Text ion 513
75666.0000 75667.0000 1.0000 [teration 256	76413.0000 76443.0000 30.0000 Iteration 512
7565.0000 75666.0000 1.0000 Fest Iteration 128	76411.0000 76413.0000 2.0000 Test Iteration 256
Iteration 128	Iteration 256
Test Iteration 64 75664.0000 75664.0000 0.0000	Test Iteration 128 76396.0000 76411.0000 15.0000
(teration 64 75663.0000 75663.0000 0.0000	Iteration 128 76395.000076396.00001.0000
75663.0000 75663.0000 0.0000	76387.0000 76395.0000 8.0000
75662.0000 75662.0000 0.0000	76387.0000 76387.0000 0.0000 Test Trention 54
75662.0000 75662.0000 0.0000 Iteration 32	76383.0000 76387.0000 4.0000 Iteration 64
75661.0000 75661.0000 0.0000 Test Iteration 16	76382.0000 76383.0000 1.0000 Test Iteration 32
75661.0009 75661.0000 0.0000 Iteration 16	Iteration 32
75661 0000 75661 0000 0 0000	Test Iteration 16 76380.0000 76382.0000 2.0000
Iteration 8 75661.0000 75661.0000 0.0000	Iteration 16 76380.0000 76380.0000 0.0000
Test learnin 4 75660,000 7560,0000 Territon 8 75661,0000 7561,0000 0.0000	1001 12022100 8 75375.0000 76380.0000 1.0000 Iteration 15 75380.0000 76380.0000 0.0000
73660.0000 75660.0000 0.0000 Tent Terration 4 73660.0000 75660.0000 0.0000 Terration 9 7561.0000 75661.0000 0.0000	76378.000076378.0000 0.0000 Test Iteration 8 76379.000076380.0000 1.0000 Iteration 16 75380.000076380.0000 0.0000
75660,0000,75660,0000 0.0000 Iteration 4 75660,0000 75660,0000 0.0000 Feet Iteration 4 75660,0000 75660,0000 0.0000 Iteration 8 75661,0000,75661,0000 0.0000	75378.000076378.0000 0.0000 Iteration 8 75378.000076378.0000 0.0000 Test Iteration 8 75379.000076380.0000 1.0000 Iteration 15 75380.000076380.0000 0.0000
75660.0000 75660.0000 0.0000 Test learning 2 75660.0000 0.0000 Test learning 7 75660.0000 0.0000 Test learning 7 75660.0000 0.0000 Test learning 7 75660.0000 0.0000 Test learning 4 75660.0000 0.0000 Test learning 7 75660.0000 0.0000	75377.0000 75377.0000 0.0000 Test Iteration 4 76378.0000 76378.0000 0.0000 Iteration 8 75378.0000 76378.0000 0.0000 Test Iteration 8 75379.0000 76380.0000 1.0000 Iteration 16 75380.0000 76380.0000 0.0000
75659.0000 7561.0000 0.0000 Test Learnian 2 75660.0000 75660.0000 0.0000 Test Learnian 2 75660.0000 75660.0000 0.0000 Test Learnian 4 75660.0000 75660.0000 0.0000 Test Learnian 4 75660.0000 75661.0000 0.0000	75177.0000 76377.0000 0.0000 Iteration 4 76377.0000 76377.0000 0.0000 Test Iteration 4 76378.0000 76378.0000 0.0000 Iteration 8 76378.0000 76378.0000 0.0000 Test Iteration 8 76379.0000 76380.0000 1.0000 Iteration 16 76380.0000 76380.0000 0.0000

78173.0000 78173.0000 Fest Iteration 2 78173.0000 78174.0000			± • •		
78173.0000 78174.0000	0.0000		79587.0000 79588.0000	1.0000	
	1.0000		79588.0000 79588.0000	0.0000	
78174.0000 78174.0000	0.0000		79589.0000 79589.0000	0.0000	
78174.0000 78175.0000	1.0000		79589.0000 79590.0000	1.0000	
[teration 8 78175.0000 78175.0000	0.0000		Iteration 256 79591.0000 79592.0000	1.0000	
Test Iteration 8 78175.0000 78176.0000	1.0000		Test Iteration 256 79592.0000 79594.0000	2.0000	
Iteration 16	0.0000		Iteration 512	2.0000	
Test Iteration 16	0.0000		Test Iteration 512	3.0000	
78176.0000 78178.0000 [teration 32	2.0000		79597.0000 79600.0000 Iteration 1024	3.0000	
78179.0000 78179.0000	0.0000		79601.0000 79606.0000 Test Iteration 1024	5.0000	
78179.0000 78182.0000	3.0000		79607.0000 79613.0000	6.0000	
78183.0000 78183.0000	0.0000		79613.0000 79625.0000	12.0000	
78183.0000 78190.0000	7.0000		79625.0000 79638.0000	13.0000	
Iteration 128 78191.0000 78191.0000	0.0000		Iteration 4096 79638.0000 79661.0000	23.0000	
Test Iteration 128	16 0000		Test Iteration 4096	36 0000	
teration 256	15.0000		Iteration 8192	20.0000	
78206.0000 78207.0000 Fest Iteration 256	1.0000		79698.0000 79734.0000 Test Iteration 8192	46.0000	
78207.0000 78236.0000	29.0000		79735.0000 79787.0000 Iteration 16384	52.0000	
78236.0000 78239.0000	3.0000		79787.0000 79879.0000	92.0000	
78239.0000 78296.0000	57.0000		79880.0000 79984.0000	104.0000	
[teration 1024 78296.0000 78302.0000	6.0000		Test Name: P000001		Class Name: Procedure
Test Iteration 1024	111.0000		CPU Time: 7.3 m Nall Time: 7.3 m	icroseconds	Iteration Count: 163
	113.0000		Test Description:	ICTOBECONDS.	
fest Name: C000002 CPU Time: 1044.9 m	aicroseconds	Class Name: Tasking	Procedure call and ret procedure is local	urn time (may	be zero if automatic inlinin
fall Time: 1044.9 m Sent Description:	sicroseconds.	Iteration Count: 1024	no parameters		
Task create and termin	ate time measuremen	nt.	Thoughting 1		
task defined and used	in procedure, no se	elect statement, no loop	79986.0000 79986.0000	0.0000	
			Test Iteration 1 79986.0000 79986.0000	0.0000	
teration 1 78783.0000 78783.0000	0.0000		Iteration 2 79986.0000 79986.0000	0.0000	
Test Iteration 1	0.0000		Test Iteration 2	0.0000	
Iteration 2	0.0000		Iteration 4	0.0000	
78784.0000 78784.0000 Test Iteration 2	0.0000		79987.0000 79987.0000 Test Iteration 4	0.0000	
78784.0000 78784.0000	0.0000		79987.0000 79987.0000	0.0000	
78784.0000 78784.0000	0.0000		79987.0000 79988.0000	1.0000	
78784.0000 78785.0000	1.0000		Test Iteration 8 79988.0000 79988.0000	0.0000	
[teration 8 78785.0000 78785.0000	0.0000		Iteration 16 79988.0000 79988.0000	0.0000	
Test Iteration 8	0.0000		Test Iteration 16	0.0000	
teration 16	0.0000		Iteration 32	0.0000	
78786.0000 78786.0000 Test Iteration 16	0.0000		79989.0000 79989.0000 Test Iteration 32	0.0000	
78786.0000 78787.0000	1.0000		79990.0000 79990.0000	0.0000	
78787.0000 78787.0000	0.0000		79990.0000 79991.0000	1.0000	
78787.0000 78789.0000	2.0000		79991.0000 79991.0000	0.0000	
teration 64 78789.0000 78790.0000	1.0000		Iteration 128 79992.0000 79993.0000	1.0000	
Cest Iteration 64	3 0000		Test Iteration 128	1 0000	
teration 120	3.0000		Iteration 256	1.0000	
78794.0000 78795.0000	1.0000		79994.0000 79996.0000	2.0000	
THE TOP ALLON 128	r		Test Iteration 256		
78795.0000 78801.0000	6.0000		Test Iteration 256 79996.0000 79999.0000 Theration 512	3.0000	
78795.0000 78801.0000 Iteration 256 78802.0000 78803.0000	1.0000		Test Iteration 256 79996.0000 79999.0000 Iteration 512 79999.0000 80002.0000	3.0000 3.0000	
(ast iteration 12 78795.0000 78801.0000 [teration 256 78802.0000 78803.0000 [est Iteration 256 78803.0000 78816.0000	1.0000 13.0000		Test lteration 256 79996.0000 79999.0000 Iteration 512 79999.0000 80002.0000 Test Iteration 512 8002.0000 80007.0000	3.0000 3.0000 5.0000	
(est lteration 128 78795.0000 78801.0000 [teration 256 78802.0000 78903.0000 (est Iteration 256 78803.0000 78816.0000 (teration 512 78816.0000 78819.0000	1.0000 13.0000 3.0000		Test Iteration 256 79996.0000 79999.0000 Iteration 512 79999.0000 80002.0000 Test Iteration 512 80002.0000 80007.0000 Iteration 1024 80007.0000 80013.0000	3.0000 3.0000 5.0000 6.0000	
Contraction 128 78795.0000 78801.0000 terration 256 78802.0000 78803.0000 Cent Iteration 256 78803.0000 78815.0000 terration 512 78816.0000 78815.0000 Cent Iteration 512	5.0000 1.0000 13.0000 3.0000		Test Iteration 256 79395.0000 73995.0000 Iteration 512 79393.0000 80002.0000 Test Iteration 512 80002.0000 80007.0000 Iteration 1024 80007.0000 80013.0000 Test Iteration 1024 80014 0000 6003 6000	3.0000 3.0000 5.0000 6.0000	
Ters: 000 7803.0000 teration 256 7802.0000 7803.0000 Terteration 256 7803.0000 78815.0000 teration 512 78816.0000 78815.0000 teration 512 78815.0000 78845.0000 teration 1024	1.0000 13.0000 3.0000 26.0000		Test Iteration 256 79995.0000 7999.0000 Iteration 512 80002.0000 8002.0000 Iteration 1024 80007.0000 8007.0000 Test Iteration 1024 80014.0000 80022.0000 Iteration 2048	3.0000 3.0000 5.0000 6.0000 8.0000	
Last learning 128 78751.0000 78801.0000 Learning 256 78803.0000 7881.000 788151000 7881.000 78816.0000 78815.0000 78816.0000 78815.0000 78815.0000 7885.0000 Learning 1024	6.0000 13.0000 3.0000 26.0000 6.0000		Test Iteration 256 79995.0000 7999.0000 Iteration 512 80002.0000 8002.0000 Iteration 1024 80007.0000 8007.0000 Test Iteration 1024 80014.0000 80022.0000 Iteration 2048 80023.0000 80036.0000 Test Iteration 2048	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000	
The second state of the se	6.000 1.0000 3.0000 26.0000 6.0000 52.0000		Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 80002.0000 Iteration 512 80002.0000 80007.0000 Iteration 1024 80017.0000 80013.0000 Test Iteration 1024 80016.0000 80022.0000 Iteration 2048 80036.0000 80052.0000 Iteration 2048	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000 17.0000	
Ters: 000 7801.000 teration 256 7802.000 7803.000 Test. 1eration 256 7803.000 78015.000 teration 512 7815.000 78015.000 Test. 100 7805.000 teration 1024 7845.000 7805.000 Test. 100 7805.000 Test.	5.000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000		Temt Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 80002.0000 Iteration 512 80002.0000 80007.0000 Iteration 1024 80014.0000 80013.0000 Iteration 2048 80035.0000 80036.0000 Iteration 4036 80035.0000 8053.0000 Iteration 4036	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000 17.0000 26.0000	
Ters: 000 7801.000 teration 256 7802.000 7803.000 Teat Teration 256 7803.000 7815.000 Teration 512 7815.000 7815.000 Teration 1024 7845.000 7845.000 Testin 1024 7851.000 7801.000 Teration 2048 7893.000 7814.0000 Testiceration 2048 7893.000 7918.0000 Testiceration 2048 7895.000 7918.0000	6.000 1.0000 13.0000 26.0000 6.0000 52.0000 11.0000		Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 80002.0000 Iteration 512 80002.0000 80013.0000 Iteration 1024 80014.0000 80022.0000 Iteration 2048 80023.0000 80036.0000 Test Iteration 2048 80036.0000 80053.0000 Iteration 4036 80036.0000 80080.0000 Iteraticn 4036 80080.0000 8015.0000	3.0000 3.0000 5.0000 8.0000 13.0000 17.0000 26.0000 35.0000	
Ters: 000 7801.000 teration 256 7802.000 7803.000 Teation 256 7803.000 7801.000 Teration 312 7815.000 78015.000 Teration 1024 7805.000 7805.000 Teration 1024 7805.000 7805.000 Teration 2048 7895.000 78018.0000 Teration 2048 7895.000 7818.0000	1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000	Class Baues Charter 13	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8002.0001 Iteration 512 80002.0000 80007.0000 Iteration 1024 80001.0000 80013.0000 Test Iteration 1024 80014.0000 8002.0000 Iteration 2048 80036.0000 8005.0000 Iteration 4095 80054.0000 80050.0000 Test Iteration 4095 80050.0000 80080.0000 Iteration 8098 80080.0000 80080.0000 Iteration 8098 80080.0000 80080.0000 Iteration 8098	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000	
Territor 128 78755.0000 78802.0000 Territor 256 78802.0000 78803.0000 Territor 256 78803.0000 78815.0000 Territor 512 78815.0000 78815.0000 Territor 1024 78851.0000 78915.0000 Territor 2048 78915.0000 78918.0000 Test 1territor 2048 78915.0000 78918.0000 Test 1territor 2048 78915.0000 78918.0000 Test 1territor 2048 78915.0000 7918.0000 Test 1territor 2048 78915.0000 7918.0000 Test 1territor 2048 78915.0000 7918.0000 Test 100004 Test 100004	1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000	Class Name: Chapter 13	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8002.0000 Iteration 512 80002.0000 80003.0000 Iteration 1024 8001.0000 80013.0000 Test Iteration 1024 80013.0000 8003.0000 Iteration 2048 80036.0000 80053.0000 Iteration 4096 80036.0000 8053.0000 Iteration 4096 80030.0000 8051.0000 Iteration 8192 80115.0000 80168.0000 Test Iteration 8192	3.0000 3.0000 5.0000 8.0000 13.0000 17.0000 26.0000 35.0000 53.0000	
Less Liesz Lion 128 78755.0000 78801.0000 Testation 256 Test 1000 78801.0000 Test 1000 78805.0000 Test 1000 78815.0000 Test 1000 78815.0000 Test 1000 78851.0000 Test 1000 78951.0000 Test 1000 78931.0000 Test 1000 7893.0000 Test 1000 7893.0000 Test 1000 7893.0000 Test 1000 7893.0000 Test 1000 78918.0000 Test 10000 78918.0000 Test 10000 78918.0000 Test 10000 78918.0000 Test 1000000 Test 1000000000000000000000000000000000000	1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 iicroseconds.	Class Name: Chapter 13 Iteration Count: 2048	Test Iteration 256 79999.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Iteration 512 80002.0000 80007.0000 Iteration 1024 90014.0000 80022.0000 Iteration 2048 80023.0000 80036.0000 Iteration 2048 80036.0000 80053.0000 Iteration 4096 80036.0000 80551.0000 Iteration 4096 80080.0000 80115.0000 Iteration 8192 80115.0000 80168.0000 Test Iteration 8192 80168.0000 80238.0000	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 70.0000	
Least lease ton 12 7875.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 78803.0000 78805.0000 78816.0000 78816.0000 78816.0000 78816.0000 78816.0000 78815.0000 78815.0000 78815.0000 78815.0000 78815.0000 78931.0000 78931.0000 78931.0000 78931.0000 78931.0000 78935.000000000000000000000000000000000000	1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 iicroseconds iicroseconds iicroseconds	Class Name: Chapter 13 Iteration Count: 2048 outs on arrays of booleans.	Test Iteration 256 79395.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Test Iteration 512 80002.0000 80007.0000 Iteration 1024 80010.0000 80013.0000 Test Iteration 204 80016.000 8002.0000 Iteration 204 80036.0000 80551.0000 Iteration 4095 80034.0000 80501.0000 Iteration 4095 80080.0000 80115.0000 Iteration 8192 80115.0000 80182.0000 Iteration 8192 80168.0000 8034.0000 Iteration 1534	3.0000 3.0000 5.0000 8.0000 13.0000 17.0000 26.0000 35.0000 70.0000 105.0000	
The set of	1.0000 1.0000 13.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 ticroseconds ticroseconds ticroseconds ticroseconds ticroseconds ticroseconds ticroseconds	Class Name: Chapter 13 Iteration Count: 2048 ons on grays of booleans. with the progme 'PACK. ad on components in a loop.	Test Iteration 256 73936.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Iteration 512 80002.0000 80007.0000 Iteration 1024 80013.0000 80013.0000 Test Iteration 2048 80035.0000 8003.0000 Iteration 4036 80035.0000 8003.0000 Iteration 4036 80035.0000 8003.0000 Iteration 4036 8005.0000 8003.0000 Iteration 8192 80115.0000 80158.0000 Iteration 8192 90168.0000 8034.0000 Iteration 16344 80343.0000 80442.0000	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 105.0000 139.0000	
Ters: 000 7801.000 teration 256 7802.000 7803.000 Test: 000 7801.000 Test: 000 7801.000 Test: 000 7801.000 Test: 000 7801.000 Test: 000 7804.000 Test: 000 7804.000 Test: 000 7804.000 Test: 1eration 1024 7803.000 7801.000 Test: 1eration 2048 7803.000 7801.000 Test: 1eration 2048 7803.000 7801.000 Test: 1eration 2048 7813.000 7801.0000 Test: 1eration 2048 Test: 000 7801.0000 Test: 1eration 2048 Test: 000 7801.0000 Test: 1eration 2048 Test: 000 7801.0000 Test: 1eration 1024 Test: 0000 7801.0000 Test: 1eration 1024 Test: 000 7801.0000 Test: 1eration 100 Test: 1eration 100 T	5.000 1.0000 13.0000 3.0000 26.0000 52.0000 11.0000 103.0000 103.0000 107.0000 107.0000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.000 107.00000 107.00000 107.00000 10	Class Name: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'DACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 80002.0000 Iteration 512 80002.0000 80007.0000 Iteration 1024 80014.0000 80022.0000 Iteration 2048 80035.0000 8003.0000 Test Iteration 2048 80035.0000 80053.0000 Iteration 4036 80035.0000 80080.0000 Test Iteration 4036 80085.0000 80080.0000 Iteration 8192 8015.0000 80080.0000 Iteration 8192 80166.0000 8031.0000 Iteration 16384 80238.0000 80343.0000 Iteration 16384 80238.0000 80343.0000 Test Iteration 16384 80343.0000 8042.0000	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 70.0000 139.0000	Class Name: Procedure
Terration 12 78755.0000 78801.0000 Terration 256 78802.0000 78801.0000 Terration 512 78810.0000 78815.0000 Terration 512 78815.0000 78815.0000 Terration 1024 78851.0000 78951.0000 Terration 1024 78951.0000 78951.0000 Test Mame: M00004 CPU Time: 449.2 m Test Description: Time to perform standa For this test the arcs For this test the oper Kterration 1 75584.0000 75584.0000	0.0000 1.0000 1.0000 3.0000 26.0000 52.0000 11.0000 103.0000 107.0000 107.0000 107.0000 0.0000	Class Name: Chapter 13 Iteration Count: 2048 cons on arrays of booleans. with the pragma 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8007.0000 Iteration 512 80002.0000 8007.0000 Iteration 1024 80014.0000 8002.0000 Iteration 2048 80035.0000 8003.0000 Iteration 2048 80036.0000 8003.0000 Iteration 4095 80053.0000 8003.0000 Iteration 4095 80080.0000 8015.0000 Iteration 8192 80168.0000 8045.0000 Iteration 8192 80168.0000 8045.0000 Iteration 16384 80238.0000 8042.0000 Iteration 16384 8033.0000 8042.0000 Test Iteration 16384 8034.0000 8042.0000 Iteration 16384 8034.0000 8042.0000 Iter Iteration 20.8 m Vall Iteration 20.8 m	3.0000 3.0000 5.0000 6.0000 13.0000 13.0000 26.0000 35.0000 53.0000 105.0000 105.0000 139.0000 139.0000	Class Name: Procedure Iteration Count: 163
The section 128 of 128	0.0000 1.0000 1.0000 3.0000 26.0000 52.0000 11.0000 103.0000 1070seconds. NCTOSeconds. NCTOSeconds. NCTOSeconds. 1070seconds. 107	Class Mame: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8002.0000 Iteration 512 80002.0000 80003.0000 Iteration 1024 80014.0000 8002.0000 Iteration 2048 80035.0000 8003.0000 Iteration 2048 80036.0000 8005.0000 Iteration 4036 80036.0000 8005.0000 Iteration 8132 8015.0000 8015.0000 Iteration 8132 8015.0000 8045.0000 Iteration 8132 80168.0000 8042.0000 Iteration 16344 80238.0000 8042.0000 Iteration 16344 80238.0000 8042.0000 Iteration 1634 8034.0000 8042.0000 Iteration 20.8 m Wall Time: 20.8 m Wall Time: 20.8 m	3.0000 3.0000 5.0000 8.0000 13.0000 17.0000 26.0000 35.0000 35.0000 105.0000 139.0000 139.0000 139.0000	Class Name: Procedure Iteration Count: 163
Least lease long 128 million 1	0.0000 1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 107.0000 107.0000 0.0000 0.0000 0.0000	Class Mame: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'PACK.' ed on components in a loop.	Test Iteration 256 79995.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Iteration 1024 10002.0000 80007.0000 Iteration 1024 90014.0000 80022.0000 Iteration 2048 80023.0000 80036.0000 Test Iteration 2048 80036.0000 80036.0000 Iteration 4095 80080.0000 80115.0000 Iteration 4095 80080.0000 80115.0000 Iteration 8192 80115.0000 80168.0000 Test Iteration 1034 80238.0000 80343.0000 Test Iteration 1634 80343.0000 80422.0000 Test Description: Procedure call and ret Compare to P000005 10 parameters in Iteration	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 105.0000 139.0000 139.0000 icroseconds. urn time measus	Class Name: Procedure Iteration Count: 163 rement
Least lease long 128 18755.0000 78801.0000 Least long 256 18802.0000 78801.0000 Test Least long 256 18808.1000 78801.0000 18808.1000 78819.0000 18818.0000 78819.0000 18818.0000 78851.0000 18818.0000 78931.0000 18818.0000 78918.0000 18815.0000 78918.0000 18815.0000 78918.0000 Cest Least long 2048 78913.0000 78918.0000 Cest Least long 2048 78915.0000 78918.0000 Cest Least long 2048 78915.0000 79918.0000 Cest Mame: M00004 78915.0000 79918.0000 Cest Least long 2048 78918.0000 79918.0000 Cest Least long 2048 78918.0000 79918.0000 Cest Least long 2048 For this test the oper Cest Least long 27584.0000 Cest Least 27584.0000 Cest Least 27584.0000	0.0000 1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 107.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Chapter 13 Iteration Count: 2048 Ons on arrays of booleans. with the pragma 'PACK,' ad on components in a loop.	Test Iteration 256 79395.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Test Iteration 512 80007.0000 8007.0000 Test Iteration 1024 80007.0000 80013.0000 Test Iteration 2048 80023.0000 80036.0000 Iteration 2048 80036.0000 80052.0000 Iteration 4095 80036.0000 80652.0000 Iteration 4095 80080.0000 80115.0000 Iteration 8192 80168.0000 80482.0000 Test Iteration 1634 80238.0000 80482.0000 Test Heration 1634 80343.0000 80482.0000 Test Heration 1634 80343.0000 Test Hera	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 105.0000 139.0000 139.0000 1070econds. Urn time measur	Class Name: Procedure Iteration Count: 163 rement
Least lease long 129 78755.0000 78801.0000 Testation 256 78802.0000 78801.0000 Testation 256 78803.0000 78816.0000 Testation 512 78816.0000 78815.0000 Testation 1024 78815.0000 78951.0000 Test Lease long 2048 78915.0000 79018.0000 Test Lease 100 78915.0000 79584.0000 Test Lease 100 Time to perform stands For this test the oper Terration 1 75584.0000 75584.0000 Test Lease 100 Test	0.0000 1.0000 13.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 103.0000 10000 0.0000 0.0000 0.0000 0.0000	Class Name: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragme 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8007.0000 Iteration 1024 80007.0000 8003.0000 Test Iteration 1024 80017.0000 8002.0000 Iteration 2048 80022.0000 8002.0000 Test Iteration 2048 10036.0000 8008.0000 Iteration 4096 80054.0000 8008.0000 Iteration 4096 80080.0000 80115.0000 Iteration 8192 80115.0000 80148.0000 Iteration 1892 80168.0000 80482.0000 Iteration 1634 80238.0000 80482.0000 Test Rams: P000010 CPU Time: 20.8 m Wall Time: 20.8 m Wall Time: 20.8 m Wall Time: 20.8 m Netton 11	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 35.0000 105.0000 139.0000 139.0000 105.0000 139.0000	Class Name: Procedure Iteration Count: 163 rement
The section 12 of	6.000 1.0000 1.0000 3.0000 26.0000 52.0000 11.0000 103.0000 10.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'DACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 80002.0000 Iteration 1024 80007.0000 80003.0000 Iteration 1024 80014.0008 80002.0000 Iteration 2048 80035.0000 8003.0000 Test Iteration 2048 80035.0000 80030.0000 Test Iteration 4036 80036.0000 80040.0000 Test Iteration 8192 80135.0000 80150.0000 Iteration 8192 80135.0000 80150.0000 Iteration 8192 80135.0000 8042.0000 Test Iteration 1634 80343.0000 8042.0000 Test Iteration 1634 80343.0000 8042.0000 Test Hems: P00010 CPU Time: 20.6 m Well Time: 20.6 m Well Time: 20.6 m Iteration 1 80455.0000 80455.0000 Test Iteration 1 80455.0000 Test Iteration 1 80455.0000 Test Iteration 1 80455.0000 Test Iteration 1 80455.0000 Test Iteration 1 80455.0000 Test Iteration 1 80455.0000 Test Iteration 1 8045.0000 Test Iteration 1 8045.0000 Test Iteration 1 8045.0000 Test Iteration 1 8045.0000	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 105.0000 139.0000 139.0000 139.0000 105.0000	Class Name: Procedure Iteration Count: 163 rement
The section 12 of	6.000 1.0000 1.0000 3.0000 26.0000 52.0000 11.0000 103.0000 sicroseconds icroseconds icroseconds icroseconds icroseconds 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Chapter 13 Iteration Count: 2048 ons on srrays of booleans. with the pragma 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8002.0000 Iteration 1024 80007.0000 8002.0000 Iteration 1024 80014.0000 80022.0000 Iteration 2048 80035.0000 80080.0000 Test Iteration 2048 80035.0000 80080.0000 Test Iteration 4036 80035.0000 80080.0000 Iteration 8132 80135.0000 80080.0000 Iteration 8132 80165.0000 80080.0000 Test Iteration 16384 80238.0000 80343.0000 Test Iteration 16384 80238.0000 8042.0000 Iteration 16384 80238.0000 8042.0000 Test Iteration 16384 80238.0000 8042.0000 Test Iteration 16384 80238.0000 8042.0000 Iteration 16384 80238.0000 8042.0000 Iteration 16384 80238.0000 10000 Iteration 1 MTE Iteration 1 MTE Iteration 1 MTE Iteration 1 MTE Iteration 1 MTE Iteration 1 80455.0000 80455.0000 Iteration 1 80455.0000 80455.0000	3.0000 3.0000 5.0000 6.0000 13.0000 13.0000 26.0000 26.0000 35.0000 53.0000 105.0000 105.0000 139.0000 139.0000 139.0000 139.0000 0.0000 0.0000	Class Name: Procedure Iteration Count: 163 rement
Last learning 129 Tarss. 0000 78801.0000 Learning 256 7880.0000 78801.0000 7880.1000 78801.0000 7880.1000 78815.0000 7881.6.0000 78815.0000 rest Learning 1024 78815.0000 7891.0000 rest Learning 2048 78915.0000 78914.0000 Creation 449.2 m For this Lest the arra For this Lest the arra For this Lest the creation 1 7554.0000 75544.0000 Creation 2 7554.0000 75544.0000 Creation 7554.0000 Tiste fearation 4 75554.0000 7555.0000 Creation 75555.0000 Creation 7555.0000 Creation 7555.00000 Creation 7555.0000 Cr	0.0000 1.0000 1.0000 3.0000 26.0000 52.0000 11.0000 103.0000 103.0000 107.0000 107.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Hame: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'PACK.' ad on components in a loop.	Test Iteration 256 79995.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Iteration 1024 10002.0000 80007.0000 Iteration 1024 90014.0000 80022.0000 Iteration 2048 80023.0000 80035.0000 Iteration 2048 80036.0000 80053.0000 Iteration 4095 80080.0000 80115.0000 Iteration 8192 80115.0000 80150.0000 Iteration 8192 80168.0000 80450.0000 Test Iteration 1634 80238.0000 80450.0000 Test Iteration 1094 80343.0000 80450.0000 Test Iteration 1094 80343.0000 80450.0000 Test Iteration 1094 80343.0000 80450.0000 Iteration 1094 80343.0000 80450.0000 Iteration 1094 80343.0000 80450.0000 Iteration 1000 10 parameters, in INTE Iteration 1 80485.0000 80450.0000 Test Iteration 1 80485.0000 80450.0000 10 parameters, in INTE 10	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 105.0000 105.0000 139.0000 139.0000 icroseconds. urn time measur GER 0.0000 0.0000	Class Name: Procedure Iteration Count: 163 rement
Least lease Lon 128 7875.000 78801.0000 Test ion 256 Test ion 256 78802.0000 78801.0000 Test is concerned in 0000 78805.0000 78801.0000 78816.0000 78815.0000 78816.0000 78851.0000 Test is ration 1024 78815.0000 78914.0000 Test 16 78914.00000 Test 16 78914.00000 Test 16 78914.000	0.0000 1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 103.0000 107.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Mame: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'PACK.' ed on components in a loop.	Test Iteration 256 79395.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Test Iteration 512 80007.0000 8007.0000 Test Iteration 1024 80007.0000 80013.0000 Test Iteration 204 80023.0000 8003.0000 Iteration 2048 80036.0000 8005.0000 Iteration 4095 80080.0000 8065.0000 Iteration 8192 80168.0000 80482.0000 Iteration 1634 80238.0000 80482.0000 Test Hearst 1001 1teration 1634 80343.0000 80482.0000 Test Hearst 1001 Fort Hearst 1001 Procedure call and ret Compare to 900085.0000 Iteration 1 80485.0000 80485.0000 Iteration 1 80485.0000 80485.0000 Iteration 1 80485.0000 80485.0000 Iteration 1 80485.0000 80485.0000 Iteration 2 80485.0000 80485.00000 Iteration 2 80485.0000 80485.	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000 17.0000 26.0000 35.0000 105.0000 105.0000 105.0000 1070seconds. urn time measus GER 0.0000 0.0000 0.0000	Class Name: Procedure Iteration Count: 163 rement
Least Lease Lon 128 7875.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 78802.0000 7881.0000 7885.0000 7885.0000 7885.0000 7885.0000 7885.0000 7885.0000 7885.0000 7885.0000 7885.0000 7885.0000 78915.0000 79018.0000 100000 100000 1000000 10000000 1000000	0.0000 1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 103.0000 103.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Chapter 13 Iteration Count: 2048 ons on grays of booleans. with the progme 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7999.0000 Iteration 512 80002.0000 80002.0000 Test Iteration 512 80002.0000 80007.0000 Iteration 1024 80007.0000 80032.0000 Fest Iteration 204 80023.0000 80032.0000 1teration 4095 80036.0000 80052.0000 Iteration 4095 80036.0000 80115.0000 Iteration 4095 80080.0000 80115.0000 Iteration 8192 80115.0000 80169.0000 Iteration 1892 80168.0000 80482.0000 Test Hearst P000010 CPU Time: 20.8 m Wall Time: 20.8 m Wall Time: 20.8 m Wall Time: 20.8 m 1teration 1 80485.0000 80485.0000 Test Hearst ion 1 80485.0000 80485.0000 Test Iteration 1 80485.0000 80485.0000 Test Iteration 2 10 parameters, in INTE 11 s0485.0000 80485.0000 Test Iteration 2 80485.0000 80485.0000 10 parameters, in INTE 11 s0485.0000 80485.0000 10 parameters 4 80346.0000 80485.0000 10 parameters 4 80485.0000 8048	3.0000 3.0000 5.0000 6.0000 13.0000 17.0000 26.0000 35.0000 53.0000 105.0000 105.0000 139.0000 139.0000 107.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Procedure Iteration Count: 163 rement
Least Los 128 128 128 128 128 128 128 128 128 128	<pre>b.0000 1.0000 1.0000 3.0000 26.0000 6.0000 52.0000 11.0000 10.0000 bicroseconds icroseconds icroseconds crobeconds 0.0000 0</pre>	Class Name: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 80002.0000 Test Iteration 512 80002.0000 80007.0000 Iteration 1024 8001.0000 80022.0000 Test Iteration 2048 80022.0000 80022.0000 Test Iteration 2048 80022.0000 80082.0000 Test Iteration 4055 80054.0000 80082.0000 Iteration 8192 80155.0000 80155.0000 Iteration 1892 80156.0000 8042.0000 Test Iteration 1892 80168.0000 80482.0000 Test Rems: P000010 CPU Time: 20.8 m Wall Time: 20.8 m Wall Time: 20.8 m 8045.0000 80485.0000 Iteration 1 80485.0000 80485.0000 Test Iteration 2 80485.0000 80485.0000 Test Iteration 2 8048	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000 17.0000 26.0000 35.0000 35.0000 139.0000 139.0000 139.0000 139.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Procedure Iteration Count: 163 rement
Least lease long 128 78755.0000 78801.0000 Teast lease long 78801.0000 Teast lease long 78801.0000 Teast lease long 78816.0000 Teast lease long 78918.0000 Teast lease 1000 78914.0000 Teast lease 1000 78914.0000 Teast lease 1000 78918.0000 Teast lease 1000 78984.0000 Teast lease 1000 78984.0000 Teast least long 2 75584.0000 78584.0000 Teast least long 3 75584.0000 78584.0000 Teast least long 3 75585.0000 7858.0000 Teast least long 3 75585.0000 75585.0000 Teast least long 3 75585.0	5.000 1.000 1.000 3.000 26.000 52.000 52.000 10.000 10.000 10.000 10.000 0.00000 0.00000 0.0000 0.0000 0.0000 0.00000 0.00000000	Class Name: Chapter 13 Iteration Count: 2048 ons on arrays of booleans. with the pragma 'DACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8007.0000 Iteration 1024 80007.0000 8007.0000 Iteration 1024 80007.0000 8007.0000 Iteration 2048 80035.0000 8003.0000 Test Iteration 2048 80035.0000 8003.0000 Iteration 4036 80035.0000 80040.000 Test Iteration 1036 80035.0000 80040.000 Test Iteration 1036 80035.0000 8015.0000 Iteration 8192 80168.0000 8045.0000 Test Iteration 1634 80343.0000 80482.0000 Test Heast Point 10 Fertion 1034 80343.0000 80482.0000 Test Heast Iteration 1 8045.0000 8045.0000 Iteration 1 8045.0000 8045.0000 Iteration 1 8045.0000 8045.0000 Test Iteration 2 80485.0000 8045.0000 Test Iteration 4 80486.0000 8045.00000 Test Iteration 4 80486.0000 8045.00000	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000 13.0000 26.0000 35.0000 53.0000 105.0000 105.0000 105.0000 105.0000 0.0000 0.0000 0.0000 0.0000 0.0000	Class Name: Procedure Iteration Count: 163 remont
Last learning last last last last last last last last	5.000 1.000 1.000 3.000 26.000 52.000 1000 10.000 10.000 10.000 0.00000 0.00000 0.00000 0.00000 0.00000000	Class Name: Chapter 13 Iteration Count: 2048 ons on srrays of booleans. with the pragma 'PACK.' ad on components in a loop.	Test Iteration 256 79395.0000 7939.0000 Iteration 512 80002.0000 8002.0000 Iteration 1024 80007.0000 8002.0000 Iteration 1024 80014.0000 80022.0000 Iteration 2048 80035.0000 80080.0000 Test Iteration 2048 80035.0000 80080.0000 Test Iteration 4036 80035.0000 80080.0000 Test Iteration 8132 80135.0000 80080.0000 Iteration 8132 80135.0000 80080.0000 Test Iteration 16384 80234.0000 8042.0000 Iteration 16384 80234.0000 8042.0000 Test Iteration 16384 80234.0000 8042.0000 Test Iteration 16384 80235.0000 8042.0000 Test Iteration 16384 80235.0000 8042.0000 Iteration 1 8045.0000 Test Iteration 2 8045.0000 8045.0000 Test Iteration 2 8045.0000 8045.0000 Test Iteration 2 8045.0000 8045.0000 Iteration 4 8045.0000 8045.0000 I	3.0000 3.0000 5.0000 6.0000 8.0000 13.0000 17.0000 26.0000 26.0000 105.0000 105.0000 139.0000 139.0000 139.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000000	Class Mamer Procedure Iteration Count: 163 remont

10	8
80487.0000 80488.0000 1.0000	Iteration 64
Test Iteration 16 80488.0000 80489.0000 1.0000	37491.0000 37492.0000 1.0000 Iteration 128
Iteration 32 80489.0000 80489.0000 0.0000	37499.0000 37500.0000 1.0000 Iteration 256
Test Iteration 32 80489.0000 80491.0000 2.0000	Iteration 512
80491.0000 80491.0000 0.0000	Iteration 1024
Test Iteration 64 80492.0000 80495.0000 3.0000	Test Maney C000001 Class Maney Testing
80495.0000 80496.0000 1.0000	CPU Time: 1005.9 microseconds Wall Time: 1005.9 microseconds
80496.0000 80502.0000 6.0000	Test Description: Task create and terminate measurement
80502.0000 80504.0000 2.0000 Test Iteration 256	with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop,
80504.0000 80517.0000 13.0000 Iteration 512	
80517.0000 80520.0000 3.0000 Test Iteration 512	Iteration 1 37718.0000 37718.0000 0.0000
80520.0000 80545.0000 25.0000 Iteration 1024	Iteration 2 37718.0000 37718.0000 0.0000
80346.0000 80551.0000 5.0000 Test Iteration 1024	Iteration 4 37719.0000 37719.0000 0.0000
80551.0000 80602.0000 51.0000 Iteration 2048	Iteration 6 37719.0000 37719.0000 0.0000
80602.0000 80613.0000 11.0000 Test Iteration 2048 Rest Gang estit once 101	107720.0000 37720.0000 0.0000
	37723.0000 37723.0000 0.0000
CPU Time: 439.5 microseconds	37726.0000 37727.0000 1.0000
Test Description: Minimum renderwore, entry call and raturn time	37734.0000 37735.0000 1.0000
1 task 1 entry , task inside procedure	37749.0000 37751.0000 2.0000 Iteration 512
	37779.0000 37782.0000 3.0000 Terration 1024
Iteration 1 \$0718.0000 80718.0000 0.0000	37839.0000 37845.0000 6.0000
Test Iteration 1 \$0718.0000 \$0718.0000 0.0000	Test Name: C000002 Class Name: Tasking CPU Time: 1054.7 microseconds
Iteration 2 80718.0000 80718.0000 0.0000	Nall Time: 1054.7 microseconds. Iteration Count: 1024 Test Description:
Test Iteration 2 80719 0000 - 5719,0000 5,0000	Task create and terminate time measurement. with one task, no entries when task is in a procedure,
Iteration 4 80719.0000 80719.0000 0.0000	task defined and used in procedure, no select statement, no loop
Test Iteration 4 80719.0000 80720.0000 1.0000	
Iteration # #0720.0000 80720.0000 0.0000	Iteration 1 38327.0000 38327.0000 0.0000
Test Iteration 9 00720.0000 80721.0000 1.0000	1007 2 38328.0000 38328.0000 0.0000
10 10 10 10 10 10 10 10 10 10 10 10 10 1	1 termition 4 38328.0000 38328.0000 0.0000
1080 1080 10 15 80722.0000 #0724.0000 2.0000	38328.0000 38329.0000 1.0000
100724.0000 80724.0000 0.0000	38329.0000 38329.0000 0.0000
#0724.0000 #0728.0000 4.0000	38330.0000 38330.0000 0.0000
80728.0000 80729.0000 1.0000	38332.0000 38333.0000 1.0000
80729.0000 60736.0000 7.0000	38336.0000 38337.0000 1.0000 Terration 256
80737.0000 80737.0000 0.0000 Test Iteration 128	38343.0000 38345.0000 2.0000 Iteration 512
\$3738.0000 80752.0000 14.0000 Iteration 256	38358.0000 38361.0000 3.0000 Iteration 1024
80753.0000 80754.0000 1.0000 Test Iteration 256	38387.0000 38392.0000 5.0000 Iteration 2048
80754.0000 80784.0000 30.0000 Iteration 512	38444.0000 38455.0000 11.0000
80784.0000 80787.0000 3.0000 Test Iteration 512	Test Name: H000004 Class Name: Chapter 13 CPU Time: 449.2 microseconds
80787.0000 80847.0000 60.0000 Iteration 1024	Wall Time: 449.2 microseconds. Iteration Count: 2048 Test Description:
00147.0000 00533.0000 5.0000 Test Iteration 1024	Time to perform standard boolean operations on arrays of booleans. For this test the arrays are NOT PACKED with the pragma 'PACK.'
Test Manage TODADA	for this test the operations are performed on components in a toop.
CPU Time: 551.8 sicroseconds Wall Time: 551.8 sicroseconds Teration Count: 1024	Iteration 1 39118.0000 39118.0000 0.0000
Test Description:	Iteration 2 3919.0000 39119.0000 0.0000
One tasks active, two entries, tasks in a package using melect statement	Iteration 4 39119.0000 39119.0000 0.0000
,	Iteration 1 39119.0000 39119.0000 0.0000
Test Name: A000090 Clock resolution measurement running	Iteration 16 39119.0000 39120.0000 1.0000
Test Description: Determine clock resolution using second differences	Iteration 32 39120.0000 39120.0000 0.0000
of values returned by the function CPU_Time_Clock.	Iteration 64 39120.0000 39121.0000 1.0000
Number of sample values is 7000 Clock Resolution = 1.0000000000000 seconds.	Iteration 128 39121.0000 39122.0000 1.0000
Clock Resolution (average) = 1.0000000000000 seconds. Clock Resolution (variance) = 0.000000000000 seconds.	Iteration 256 39123.0000 39125.0000 2.0000
	1000 112 12 12 12 12 12 12 12 12 12 12 12 12
Test Mame: A000091 Class Mame: Composite 1.2000 is time in milliseconds for one Dhrystone	100711000 1024 39133.0000 39138.0000 5.0000
rest vescription: Reinhold P. Weicker's DHRYSTOWE composite benchmark	39145.0000 39156.0000 11.0000 Terretion 4006
Test Name: 2000092 Class Name: composite	39169.0000 39192.0000 23.0000 Iteration 8192
ton many, never the composite the second sec	39217.0000 39262.0000 45.0000 Tteration 16384
Average Whetstone rating : 351 INTPS	39313.0000 39403.0000 90.0000
III A A A A A A A A A A A A A A A A A A	Test Name: \$000001 Class Name: Procedure CPU Time: 7.9 microseconds
Iteration 1 37482.0000 37482.0000 0.0000	Wall Time: 7.9 sicroseconds. Iteration Count: 16384 Test Description:
Iteration 2 37483.0000 37483.0000 0.0000	Procedure call and return time (may be zero if automatic inlining) procedure is local
Iteration 4 37483.0000 37483.0000 0.0000	no parameters
Iteration 8 37484.0000 37484.0000 0.0000	Iteration 1
Iteration 16 37485.0000 37485.0000 0.0000	39508.0000 39508.0000 0.0000 Iteration 2
Thornhigh 23	29509 0000 29509 0000 0 0000
37487.0000 37488.0000 1.0000	Iteration 4

	Iteration 8
Iteration 0 39589.0000 39509.0000 0.0000	77819.0000 77819.0000 0.0000 Iteration 16
Iteration 16	77820.0000 77820.0000 0.0000
Iteration 32	77822.0000 77822.0000 0.0000
39510.0000 39510.0000 0.0000 Iteration 64	Iteration 64 77#26.0000 77#26.0000 0.0000
39511.0000 39511.0000 0.0000	Iteration 128
39512.0000 39513.0000 1.0000	Iteration 256
Iteration 256 39514.0000 39515.0000 1.0000	77847.0000 77848.0000 1.0000 Iteration 512
Iteration 512	77875.0000 77878.0000 3.0000
Iteration 1024	77932.0000 77937.0000 5.0000
39525.0000 39532.0000 7.0000 Iteration 2048	Test Name: C000001 Class Name: Tasking
39540.0000 39553.0000 13.0000	CPU Time: 996.1 microseconds
39569.0000 39594.0000 25.0000	Test Description:
Iteration 8192 39627.0000 39678.0000 51.0000	Task create and terminate measurement with one task, no entries, when task is in a procedure
Tterntion 16384 39742.0000 39844.0000 102.0000	using a task type in a package, no select statement, no loop,
	••• •••
CPU Time: 15.9 microseconds Class Name: Procedure	Iteration 1 78046.0000 78046.0000 0.0000
Wall Time: 16.5 microseconds. Iteration Count: 16384	Iteration 2 78047.0000.78047.0000 0.0000
Procedure call and return time measurement	Iteration 4
Compare to P000005 10 parameters, in INTEGER	78047.0000 78047.000 0.0000 Iteration %
	78048.0000 78048.0000 0.0000 Iteration 16
Iteration 1	78049.0000 78049.0000 0.0000
Iteration 2	78051.0000 78051.0000 0.0000
39976.0000 39976.0000 0.0000 Tteration 4	Iteration 64 78055.0000 /8056.0000 1.0000
39976.0000 39976.0000 0.0000 Theration 8	Iteration 128 78653 0000 78654 0000 1.0000
39977.0000 39977.0000 0.0000	Iteration 256
Iteration 16 39978.0000 39978.0000 0.0000	78078.0000 78080.0000 2.0000 Iteration 512
Iteration 32	78108.0000 78111.0000 3.0000
39979.0000 39979.0000 0.0000 Iteration 64	- eration 1024 /#167.0000 78172.0000 5.0000
39901.0000 39901.0000 0.0000 Iteration 128	Test Name: C000002 Class Name: Tasking
39985.0000 39985.0000 0.0000	CPU Time: 1054.7 microseconds
39992.0000 39993.0000 1.0000	Test Description:
1007ation 512 40007.0000 40009.0000 2.0000	Task create and terminate time measurement. with one task, no entries when task is in a procedure,
Iteration 1024 40035.0000 40041.0000 6.0000	task defined and used in procedure, no select statement, no loop
Iterstion 2048	Iteration 1
Test Manas, T000001	78651.0000 78651.0000 0.0000
CPU Time: 454.1 miszoneconds	74651.0000 78651.0000 0.0000
Mail Time: 454.1 microseconds. Iteration Count: 2048 Test Description:	Iteration 4 78652.0000 78652.0000 0.0000
Minimum rendezvous, entry call and return time 1 task 1 entry , task inside procedure	Iteration 8 78653.0000 78653.0000 0.0000
no select	Iteration 16 78552 0000 78554 0000 1 0000
	Iteration 32
***	78/FF 8000 78/FF 0000 0 0000
Iteration 1 40210.0000 40210.0000 0.0000	78655.0000 78655.0000 0.0000 Iteration 64
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000	78655.0000 78655.0000 0.0000 Tearation 64 78657.0000 78657.0000 0.0000 Tearation 128
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 0.0000 40210.0000 0.0000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78657.0000 0.0000 Iteration 128 78660.0000 78661.0000 1.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 Iteration 4 40211.0000 40210.0000 Iteration 4 0.0000 11.0000 Iteration 8 0.0000 11.0000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78657.0000 0.0000 Iteration 128 78660.0000 78661.0000 1.0000 Iteration 256 78668.0000 78669.0000 1.0000
Iteration 1 40210.0000 40210.0000 Iteration 2 40210.0000 40210.0000 Iteration 4 40211.0000 40211.0000 Iteration 4 40212.0000 0.0000 Iteration 8 40212.0000 0.0000 Iteration 16	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78657.0000 0.0000 Iteration 128 78660.0000 78661.0000 1.0000 Iteration 256 78668.0000 78669.0000 1.0000 Iteration 512 78688.0000 78685.0000 2.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 12 40213.0000 40213.0000 0.0000 Iteration 12 40213.0000 40213.0000 0.0000 10000	78655.0000 78655.0000 0.0000 Heration 64 78657.0000 78657.0000 0.0000 Heration 128 78660.0000 7869.0000 1.0000 Heration 512 7868.0000 7869.0000 1.0000 Heration 512 78683.0000 7865.0000 2.0000 Heration 1024
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 _00215.0000 0.0000	78655.0000 78655.0000 0.0000 Herration 64 78657.0000 78657.0000 0.0000 Herration 128 78660.0000 78691.0000 1.0000 Herration 512 78683.0000 78695.0000 2.0000 Herration 1024 78712.0000 78718.0000 6.0000 Herration 2048
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40215.0000 0.0000 Iteration 64 40219.0000 40220.0000 1.0000	78655.0000 78655.0000 0.0000 Herration 64 78657.0000 78657.0000 0.0000 Herration 128 78660.0000 78661.0000 1.0000 Herration 256 78668.0000 78659.0000 1.0000 Herration 1024 78712.0000 78781.0000 11.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 2 0.0000 Iteration 4 0.0000 Iteration 6 0.0000 Iteration 8 0.0000 Iteration 16 0.0000 Iteration 16 0.0000 Iteration 2 0.0000 Iteration 32 0.0000 Iteration 32 0.0000 Iteration 40213.0000 0.0000 Iteration 32 0.0000 Iteration 64 0.0000 Iteration 12 0.0000	78655.0000 78655.0000 0.0000 Hearting 64 78657.0000 78657.0000 1.0000 Hearting 128 78660.0000 78651.0000 1.0000 Hearting 256 78683.0000 78655.0000 1.0000 Hearting 102 78683.0000 78785.0000 2.0000 Hearting 204 78710.0000 78781.0000 11.0000 Tearting 2048 78770.0000 78781.0000 11.0000 Tearting 2048 78770.0000 78781.0000 11.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 0.0000 Iteration 4 0.0000 40210.0000 0.0000 Iteration 4 0.0000 40211.0000 0.0000 Iteration 8 40212.0000 0.0000 1 Iteration 16 0.0000 1 1 Iteration 12 0.0000 1 1 1 Iteration 14 0.0000 1 1 1 1 Iteration 14 0.0000 1 0.0000 1	78655.0000 78655.0000 0.0000 Heration 64 78657.0000 78657.0000 0.0000 Heration 128 78660.0000 78651.0000 1.0000 Heration 512 7868.0000 78655.0000 2.0000 Heration 512 78683.0000 78718.0000 6.0000 Heration 2048 78770.0000 78718.0000 11.0000 Test Name: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Nall Time: 449.2 microseconds Heration Count: 2048
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40210.0000 40211.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 0.0000 Iteration 16 40213.0000 0.0000 Iteration 32 40215.0000 0.0000 Iteration 12 402220.0000 1.0000 Iteration 236 402220.0000 0.0000 Iteration 236 402220.0000 0.0000 Iteration 236 40220.0000 0.0000 Iteration 256 40224.0000 40245.0000 Iteration 256 40224.0000 40245.0000	78655.0000 78655.0000 0.0000 Heration 64 78657.0000 78657.0000 0.0000 Heration 128 78660.0000 7869.0000 1.0000 Heration 512 7868.0000 7869.0000 2.0000 Heration 1024 78112.0000 78718.0000 1.0000 Heration 2048 78770.0000 78781.0004 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Iteration Count: 2048 Test Description:
Iteration 1 0210.0000 40210.0000 Iteration 2 0210.0000 40210.0000 40210.0000 0.0000 Iteration 4 0210.0000 40210.0000 40211.0000 0.0000 Iteration 8 0010.0000 40210.0000 40212.0000 0.0000 Iteration 8 0010.0000 Iteration 16 00000 Iteration 28 00000 Iteration 32 00000 Iteration 32 00000 Iteration 32 00000 Iteration 32 0000 Iteration 512 229.0000 Iteration 512 229.0000 0000	78655.0000 78655.0000 Iteration 64 78657.0000 78657.0000 Iteration 128 78660.0000 7869.0000 Iteration 256 7868.0000 7869.0000 Iteration 512 7868.0000 7869.0000 Iteration 512 7868.0000 7869.0000 Iteration 1024 78712.0000 78718.0000 Iteration 2048 78770.0000 7871.0000 Iteration 2449.2 Sicroseconds Iteration Count: Vall Time: 449.2 microseconds. Wall Time: 449.2 microseconds. Time to perform standard boolean operations on arrays of booleans. For this teet the arrays are WOT PACED with the prage 'PACE'.
Iteration 1	78655.0000 78655.0000 Iteration 64 78657.0000 78657.0000 Iteration 128 78660.0000 78661.0000 Iteration 256 78680.0000 78695.0000 Iteration 512 78680.0000 78695.0000 Iteration 512 78680.0000 7869.0000 Iteration 1024 78770.0000 78718.0000 Iteration 2048 78770.0000 7871.0000 Iteration 2048 78770.0007 11.0000 Test Name: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: Timestory attandard boolean operations on arrays of booleans. Por this test the arrays are MOT PACKED with the pragen 'PACK'. For this test the operations are performed on components in a loop.
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 Iteration 4 40211.0000 40211.0000 Iteration 8 40211.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 Iteration 32 40215.0000 40218.0000 Iteration 228 40219.0000 40218.0000 Iteration 228 40218.0000 4028.0000 Iteration 228 40228.0000 4028.0000 Iteration 512 40244.0000 40245.0000 Iteration 1024 40340.0000 40246.0000 Iteration 1024 40340.0000 40346.0000 Iteration 1024 40340.0000 Iteration 1024 40340.0000 40340.0000 4	78655.0000 78655.0000 Iteration 64 78657.0000 78650.0000 78657.0000 Iteration 128 78660.0000 78660.0000 7869.0000 Iteration 256 78668.0000 78683.0000 7869.0000 Iteration 512 78683.0000 78770.0000 78718.0000 Iteration 2048 6.0000 78770.0000 78710.0000 Teat ion 2048 Class Name: Chapter 13 CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: 449.2 microseconds Fost Description: Time to perform standard boolean operations on arrays of booleans. For this test the arrays are MOT PACKED with the pragen 'PACK'. For this test the operations are performed on components in a loop.
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40228.0000 0.0000 Iteration 28 40219.0000 40228.0000 0.0000 Iteration 256 40244.0000 4028.0000 1.0000 Iteration 512 40276.0000 4028.0000 1.0000 Iteration 256 40244.0000 4028.0000 1.0000 Iteration 256 40246.0000 4028.0000 1.0000 Iteration 512 40276.0000 4028.0000 1.0000 Iteration 512 40276.0000 4028.0000 Class Mame: Tasking CPU Time: 571.3 microseconds Wall Time: 571.3 microseconds Iteration Count: 1024	78655.0000 78655.0000 Iteration 64 78657.0000 78657.0000 Iteration 128 78660.0000 78651.0000 Iteration 256 78680.0000 78650.0000 Iteration 512 78683.0000 7869.0000 Iteration 512 78683.0000 7869.0000 Iteration 1024 78770.0000 78712.0000 Iteration 2048 78770.0000 7871.0000 Iteration 2048 Test Base: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds Test Description: Time test the array are MOT PACRED with the pragma "PACK." For this test the operations are performed on components in a loop. Iteration 1 79400.0000 0.0000 Iteration 2 0
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40212.0000 0.0000 Iteration 32 40215.0000 40215.0000 0.0000 Iteration 228.0000 0.0000 Iteration 228.0000 0.0000 Iteration 512 40244.0000 40228.0000 1.0000 Iteration 512 40244.000 4024.0000 1.0000 Iteration 1024 40240.0000 4025.0000 1.0000 Iteration 512.0000 1.0000 Iteration 513.0000 Iteration 513.0000 Iteration 513.0000 Iteration 513.0000 Iteration 513.0000 Iteration 513.0000 Iteration 1024 40340.0000 40346.0000 5.0000 Iteration 1024 40340.0000 40346.0000 5.0000 Iteration 1024 40340.0000 40346.0000 5.0000 Iteration 1024 40340.0000 40346.0000 5.0000 Iteration 1024 Task mane: To00084 Class Mame: Tasking CPU Time: 571.3 microseconds Iteration Count: 1024 Test Description	78655.0000 78655.0000 0.0000 Heration 64 78657.0000 78657.0000 0.0000 Heration 128 78660.0000 78661.0000 1.0000 Heration 512 78680.0000 78695.0000 1.0000 Heration 1024 78712.0000 78781.0000 1.0000 Heration 2048 78770.0000 78781.0000 11.0000 Teat Name: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: 449.2 microseconds Herations on arrays of booleans. For this test the arrays are HOT PACKED with the pragma 'PACK.' For this test the operations are performed on components in a loop. Iteration 1 79400.0000 74600.000 0.0000 Heration 2 Vanthier 4
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 32 40213.0000 40215.0000 0.0000 Iteration 6 40214.0000 40228.0000 1.0000 Iteration 225 40224.0000 40228.0000 0.0000 Iteration 325 40224.0000 40245.0000 0.0000 Iteration 512 40276.0000 40245.0000 1.0000 Iteration 512 40276.0000 40245.0000 0.0000 Iteration 513 40276.0000 40346.0000 6.0000 Teet Mame: Tonsking CPU Time: 571.3 micromeconds Mall Time: 571.4 micromeconds Mall Time: 571.5 microm	78655.0000 78655.0000 1teration 24 78657.0000 78657.0000 1teration 25 7860.0000 7869.0000 1teration 25 7860.0000 7869.0000 1teration 204 7810.0000 78781.0000 11.0000 78770.0000 78781.0004 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Iteration Count: 2048 Test Description: Time to perform standard boolean operations on arrays of booleans. For this test the arrays are BOT PACKED with the progen 'PACK.' For this test the operations are performed on components in a loop. Iteration 2 78400.0000 79400.0000 1teration 2 78400.0000 79400.0000 1teration 2 78400.0000 79401.0000 1teration 4 79400.0000 79401.0000 1teration 4 79400.0000 79401.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40212.0000 0.0000 Iteration 12 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40213.0000 0.0000 Iteration 28 40219.000 40220.0000 1.0000 Iteration 28 40219.000 40220.0000 1.0000 Iteration 28 40228.000 40245.0000 1.0000 Iteration 512 40244.000 40245.0000 1.0000 Iteration 1024 40246.0000 40245.0000 3.0000 Iteration 1024 40240.000 40245.0000 5.0000 Test Name: Tool004 Class Mame: Tasking CPU Time: 571.1 microseconds Wall Time: 571.3 microseco	78655.0000 78655.0000 Iteration 64 78657.0000 78657.0000 Iteration 128 78660.0000 7869.0000 Iteration 256 7868.0000 7869.0000 Iteration 512 7868.0000 7869.0000 Iteration 512 7868.0000 7869.0000 Iteration 1024 7812.0000 7810.0000 Iteration 2040 Iteration 2044 Class Name: Chapter 13 CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds Test Name: H00004 Class Name: Test Description: Time test the arrays are MOT PACED with the pragma 'PACE.' For this test the arrays are MOT PACED with the pragma 'PACE.' For this test the operations are performed on components in a loop. Iteration 1 79400.0000 0.0000 Iteration 2 79400.0000 0.0000 Iteration 4 79401.0000 0.0000 Iteration 5 79400.0000 0.0000 Iteration 7
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40218.0000 0.0000 Iteration 228 40218.0000 4028.0000 0.0000 Iteration 228 40228.0000 4028.0000 Iteration 512 40248.0000 4028.0000 Iteration 512 40248.0000 4028.0000 Iteration 512 40248.0000 4028.0000 Iteration 1024 40340.0000 40346.0000 6.0000 Test Mame: To00004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: To00004 Chass Mame: To0001 Test Mame: To00004 Chass Mame: To0001 Test Mame: To00004 Chass Mame: To0001 Test Mame: To00004 Chass Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: To00004 Chass Mame: To0001 Test Mame: M000090	78655.0000 78655.0000 0.0000 Iteration 64 78650.0000 78651.0000 1.0000 Iteration 128 78660.0000 7869.0000 1.0000 Iteration 256 78680.0000 7869.0000 1.0000 Iteration 512 78680.0000 78680.0000 7869.0000 2.0000 Iteration 1024 78712.0000 78770.0000 78718.0000 6.0000 Tteration 2048 78770.0000 Test test 100004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Iteration Count: 2048 Test Description: Time test the arrays are MOT PACKED with the pragen 'PACK' For this test the operations are performed on components in a loop. 1 Iteration 1 7400.0000 Iteration 1 0.000 Iteration 1 7400.0000 Iteration 1 7400.0000 Iteration 1 7400.0000 Iteration 1 7400.0000 Iteration 1<
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.000 0.0000 Iteration 8 40213.0000 40212.0000 0.0000 Iteration 12 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40212.0000 0.0000 Iteration 25 40213.0000 40228.0000 0.0000 Iteration 256 40228.0000 40245.0000 Iteration 512 40244.0000 40245.0000 Iteration 552 40244.0000 40245.0000 Iteration 1024 40340.0000 40245.0000 Iteration 1024 40340.0000 40245.0000 Iteration 571.3 microseconds Iteration Count: 1024 Test Name: T000004 CPU Time: 571.3 microseconds Iteration Count: 1024 Test Name: X00000 Clock revolution measurement running	78655.0000 78655.0000 0.0000 Heration 64 78657.0000 78657.0000 1.0000 Heration 128 78660.0000 7869.0000 1.0000 Heration 512 7868.0000 7869.0000 2.0000 Heration 1024 78712.0000 78712.0000 6.0000 Heration 2048 78770.0000 7871.0000 11.0000 Test Name: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds Wall Time: 449.2 microseconds. Iteration Count: 2048 Test Description: Time to perform standard boolean operations on arrays of booleans. Por this test the operations are performed on components in a loop. Iteration 1 79400.0000 79401.0000 0.0000 Heration 4 79401.0000 79401.0000 0.0000 Heration 1 79401.0000 79401.0000 79401.0000 Heration 1 79401.0000 79401.0000 7940000 Heration
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40215.0000 0.0000 Iteration 28 40219.0000 40228.0000 0.0000 Iteration 28 40228.0000 40285.0000 Iteration 256 40244.0000 40245.0000 Iteration 512 40244.0000 40245.0000 Iteration 1024 40340.0000 40245.0000 Iteration 1024 40340.0000 40346.0000 Class Name: Tasking CPU Time: 571.3 microseconds Iteration 2004 40245.0000 Iteration 1024 40340.0000 40346.0000 Class Name: Tasking CPU Time: 571.3 microseconds Iteration Count: 1024 Test Name: T000004 Class Name: Tasking CPU Time: 571.3 microseconds Iteration Count: 1024 Test Name: A00090 Clock resolution mensurement running Test Name: A00090 Clock resolution mensurement running Test Description:	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78651.0000 1.0000 Iteration 256 78660.0000 78650.0000 78650.0000 1.0000 Iteration 512 78680.0000 78650.0000 78650.0000 1.0000 Iteration 512 78770.0000 78770.0000 78712.0000 6.0000 Iteration 1024 78770.0000 78770.0000 78781.0000 11.0000 Test Name: 449.2 microseconds Wall Time: 449.2 microseconds Vist test the operations are performed on components in a loop. 1 Iteration 1 79400.0000 0.0000 Iteration 2 7 7400.0000 0.0000 Iteration 4 7 7 79400.0000 9401.0000 <
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40216.0000 0.0000 Iteration 28 40216.0000 40246.0000 0.0000 Iteration 28 40228.0000 40246.0000 Iteration 512 40244.0000 40246.0000 1.0000 Iteration 1024 40340.0000 40346.0000 6.0000 Teet Mame, T000004 Class Mame: Tasking CPU Time: 571.3 micromeconds Hall Time; 571.3 micromeconds Hall Time; 571.3 micromeconds Hall Time; 571.3 micromeconds Test Mame, A00090 Clock resolution mensurement running Test Mame, A00090 Clock resolution mensurement running Test Description:	78655.0000 78655.0000 Iteration 64 78657.0000 78657.0000 Iteration 128 78660.0000 7869.0000 Iteration 256 78660.0000 7859.0000 Iteration 512 78680.0000 7855.0000 Iteration 512 78680.0000 7855.0000 Iteration 512 78670.0000 7855.0000 Iteration 2000 78670.0000 78512.0000 Iteration 2000 78672.000 78672.000 Iteration 2000 78672.000 78672.000 Iteration 1 78672.000 78672.000 Iteration 1 78400.0000 79400.0000 1teration 1 78401.0000 78401.0000 1teration 1 78401.0000 78401.0000 1teration 1 78402.0000 78401.0000 1teration 1 78401.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40213.0000 0.0000 Iteration 32 40213.000 40215.0000 0.0000 Iteration 40 40215.000 40228.0000 1.0000 Iteration 256 40224.0000 40228.0000 0.0000 Iteration 256 40224.0000 40245.0000 0.0000 Iteration 512 40275.0000 40245.0000 1.0000 Iteration 512 40276.0000 40245.0000 0.0000 Terat Mame: Tobula 4 40340.0000 40346.0000 6.0000 Test Mame: Tasking Class Mame: Tasking Club 1 Test Description: Task entry call and return time measured One tasks entiwe, two entries, tasks in a package using select statement Test Mame: Clock resolution using second differences of values returned by the function CPU_Time_Clock. Mumber of sample values is 7000 Clock Resolution = 1.00000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78661.0000 1.0000 Iteration 226 78660.0000 Iteration 236 1.0000 Iteration 23661.0000 1.0000 Iteration 236 1.0000 Iteration 232 2.0000 Iteration 102 11.0000 Iteration 242 2.0000 Iteration 242.2 2.0000 Iteration 242.2 2.0000 Iteration 242.2 2.0000 Iteration 2449.2 2.0000 Iteration 242.2 2.0000 Iteration 143.2 2.0000 Iteration 1.0000 740.0000 Time to perform standard boolean operations on arrays of booleans. Por this test the arrays are MOT PACRED with the progen 'PACK.' Por this test the operations are performed on components in a loop. 1.0000 Iteration 1 7400.0000 7400.0000 7400.0000 7401.0000 0.0000 Iteration 3 740
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40215.0000 0.0000 Iteration 32 40215.0000 40225.0000 1.0000 Iteration 225 4024.0000 40225.0000 0.0000 Iteration 325 4024.0000 40245.0000 0.0000 Iteration 312 40375.0000 40245.0000 1.0000 Iteration 312 40376.0000 40245.0000 0.0000 Teat Mame: Tasking CPU Time: 571.3 microseconds Mail Time: 571.5 microseconds Mail Tim	78655.0000 78655.0000 0.0000 Herration 64 78657.0000 78657.0000 0.0000 Herration 128 78660.0000 7869.0000 1.0000 Herration 1024 7811.0000 7811.0000 Teat Rame: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Herration Count: 2048 78770.0000 78781.0001 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Iteration Count: 2048 Test Description: Time to perform standard boolean operations on arrays of booleans. For this test the arrays are BOT PACKED with the progem 'PACK.' For this test the operations are performed on components in a loop. Iteration 1 78400.0000 79401.0000 0.0000 Iteration 8 79401.0000 79401.0000 0.0000 Iteration 7401.0000 0.0000 Iteration 7401.0000 0.0000 Iteration 16 79401.0000 79401.0000 0.0000 Iteration 7401.0000 0.0000 Iteration 7402.0000 0.0000 Iteration 7402.0000 0.0000 Iteration 7402.0000 0.0000 Iteration 7402.0000 0.0000 Iteration 7500.0000 0.0000 Iteration 7500.0000 0.0000 Iteration 50 7400.0000 75000 0.0000 Iteration 256 7400.0000 75000 0.0000 Iteration 50 Iteration 51 Iteration 51 Itera
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40212.0000 0.0000 Iteration 32 40215.0000 40213.0000 0.0000 Iteration 32 40215.0000 40220.0000 1.0000 Iteration 25 40218.0000 40220.0000 1.0000 Iteration 25 40228.0000 0.0000 Iteration 25 40228.0000 1.0000 Iteration 25 40228.0000 1.0000 Iteration 25 40245.0000 40245.0000 1.0000 Iteration 25 40245.0000 40245.0000 1.0000 Iteration 32 40340.0000 40345.0000 5.0000 Test Mame: T000004 Class Mame: Tasking CPU Time: 571.13 microseconds Wall Time: 571.13 microseconds Mall Time: 571.13 microseconds Mall Time: 571.50 microseconds	78655.0000 78655.0000 0.0000 Heration 64 78657.0000 78657.0000 0.0000 Heration 128 78660.0000 7869.0000 1.0000 Heration 1024 7810.0000 7811.0000 2.0000 Heration 1024 78170.0000 7811.0000 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Tteration Court: 2048 78770.0000 7871.0004 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Tteration Court: 2048 Test Name: H000004 Class Name: Chapter 13 CPU Time: 449.2 microseconds. Iteration Court: 2048 Test Secret S
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40212.0000 0.0000 Iteration 12 40213.000 40213.0000 0.0000 Iteration 32 40215.000 40213.0000 0.0000 Iteration 228 40219.000 40228.0000 1.0000 Iteration 228 40228.000 40245.0000 1.0000 Iteration 312 40228.000 40245.0000 1.0000 Iteration 312 40228.000 40245.0000 1.0000 Iteration 312 40228.000 40245.0000 5.0000 Teration 312 40245.0000 40245.0000 Iteration 312 40245.0000 40245.0000 Iteration 312 40245.0000 40245.0000 Iteration 312 40245.0000 40245.0000 Iteration 312 40245.0000 40245.0000 Iteration 312 40245.0000 40245.0000 Iteration 312 40245.0000 Iteration 312 Iteration Count: 1024 Test Mame: A000090 Clock resolution measurement running Test Mame: A000090 Clock resolution sing second differences of values returned by the function CPU_Time_Clock. Momber of sample values is 7000 Clock Resolution (average) = 1.000000000000000000000 seconds. Clock Resolution (average) = 1.000000000000000000000 seconds. Clock Resolution (average) = 0.0000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78651.0000 1.0000 Iteration 255 78660.0000 Iteration 256 78660.0000 78680.0000 7869.0000 1.0000 Iteration 512 78680.0000 78680.0000 7869.0000 1.0000 Iteration 1024 7812.0000 7812.0000 78718.0000 6.0000 Iteration 2048 Class Name: There: 445.2 sicroseconds. Test Name: H00004 Class Name: Chapter 13 CPU Time: 445.2 sicroseconds. Iteration Count: 2048 Test Name: H000004 Class Name: Chapter 13 CPU Time: 445.2 sicroseconds. Iteration Count: 2048 Test test the aperstions are performed on components in a loop. for this test the aperstions are performed on components in a loop. Teration 1 79400.0000 0.0000 fteration 2 79400.0000 0.0000 0.0000 fteration 32
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 16 40213.0000 40213.0000 0.0000 Iteration 32 40215.000 40212.0000 1.0000 Iteration 28 40214.0000 4024.0000 1.0000 Iteration 28 40228.0000 4024.0000 Iteration 32 40228.0000 4024 40228.0000 4024 40248.0000 40245.0000 Iteration 312 40246.0000 40245.0000 Iteration 322 4026.0000 40245.0000 Iteration 322 4026.0000 4024 40240.0000 4024 40340.0000 4024 40340.0000 4024 40340.0000 4024 1024 Test Mame: 700004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: 700004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: 700004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: 7000000 Clock resolution measurement running Test Mame: A000090 Clock resolution measurement running Test Mame: A000090 Clock Resolution (average) = 1.00000000000000000000000000000000000	78655.0000 78655.0000 Iteration 64 78657.0000 78657.0000 Iteration 128 78660.0000 78651.0000 Iteration 256 78660.0000 78650.0000 Iteration 512 78680.0000 7869.0000 Iteration 512 78680.0000 7869.0000 Iteration 1024 78170.0000 7871.0000 Teast Name: H00004 Class Name: Chass Name: CPU Time: 449.2 Sicroseconds Iteration Count: Wall Time: 449.2 sicroseconds Wall Time: 449.2 sicroseconds Test Description: Times test the arrays are BOT PACKED with the prages 'PACK' For this test the operations are performed on components in a loop. Iteration 2 7400.0000 7400.0000 0.0000 Iteration 4 7400.0000 7400.0000 0.0000 Iteration 512 7400.0000 7400.000 0.0000 Iteration 12 0
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40213.0000 0.0000 Iteration 32 40215.0000 40228.0000 0.0000 Iteration 228 40228.0000 4028.0000 0.0000 Iteration 228 40228.0000 4028.0000 Iteration 312 40228.0000 4028.0000 Iteration 312 40248.0000 4028.0000 Iteration 312 40248.0000 4028.0000 Iteration 312 40248.0000 4024 40340.0000 40248.0000 Iteration 1024 40340.0000 40248.0000 Iteration 1024 40340.0000 40346.0000 Iteration 1024 Test Mame: T000004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: 571.3 microseconds. Iteration Count: 1024 Test Mame: A000090 Clock resolution mensurement running Test Mame: A000900 Clock resolution mensurement running Test Mame: A000900 Clock resolution is 7000 Clock Resolution = 1.00000000000000000 seconds. Clock Resolution = 1.0000000000000000 seconds. Clock Resolution = 1.000000000000000 seconds. Clock Resolution (variance) = 0.00000000000000 seconds. Clock Resolution (variance) = 0.0000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78650.0000 78651.0000 1.0000 Iteration 256 78660.0000 78650.0000 78650.0000 1.0000 Iteration 256 78680.0000 78680.0000 7869.0000 1.0000 Iteration 512 78680.0000 78770.0000 78718.0000 6.0000 Iteration 2040 Class Name: 78770.0000 78710.0000 11.0000 Test Name: H00004 Class Name: Chapter 13 CPU Time: 448.2 microseconds Iteration Count: 2048 Test Description: Tise to perform standard boolean operations on arrays of booleans. Por this test the aperations are performed on components in a loop. Iteration 1 74400.0000 79400.0000 0.0000 Iteration 2 Iteration 2 78400.0000 0.0000 Iteration 2 78400.000 Iteration 16 7400.0000 0.0000 Iteration 2 78400.000 Iteration 2 78400.0000 0.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40213.0000 0.0000 Iteration 32 40215.0000 40213.0000 0.0000 Iteration 32 40215.0000 40228.0000 0.0000 Iteration 128 40228.0000 40228.0000 Iteration 256 40228.0000 40245.0000 Iteration 512 40246.0000 40245.0000 Iteration 512 40246.0000 40245.0000 Iteration 512 40246.0000 40245.0000 Iteration 1024 40340.0000 40245.0000 Iteration 1024 40340.0000 40245.0000 Iteration 1024 Test Name: T000004 Class Name: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Name: A000090 Clock resolution measurement running Test Name: A00090 Clock resolution measurement running Test Name: A00090 Clock Resolution (Purze) = 1.0000000000000000 seconds. Clock Resolution (Purze) = 1.00000000000000000000000000000000000	78655.0000 78655.0000 Iteration 64 78650.0000 78651.0000 Iteration 128 78660.0000 78650.0000 Iteration 255 78680.0000 78650.0000 Iteration 312 78680.0000 7855.0000 78680.0000 7855.0000 78680.0000 7855.0000 78680.0000 7855.0000 78680.0000 7855.0000 78680.0000 78761.0000 10000 78781.0000 78678.0000 78781.0000 78678.0000 78781.0000 78678.0000 78781.0000 78678.0000 78781.0000 78678.0000 78781.0000 78608.0000 78781.0000 78608.0000 78408.0000 78608.0000 78408.0000 78608.0000 78408.0000 78608.0000 78408.0000 78608.0000 78408.0000 78608.0000 78408.0000 78608.0000 78408.0000 78608.0000 78408.0000 <td< td=""></td<>
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40212.0000 40212.0000 0.0000 Iteration 32 40215.0000 40215.0000 0.0000 Iteration 6 40214.0000 40215.0000 0.0000 Iteration 226.0000 1.0000 Iteration 226.0000 0.0000 Iteration 312 40226.0000 40245.0000 0.0000 Iteration 312 40276.0000 40245.0000 1.0000 Iteration 1024 40340.0000 40346.0000 6.0000 Test Mame: Tobusconds Mail Time: 571.3 micromeconds Mail Time: 571.0 micromeconds Mail Time: 57000 Clock Resolution missurement running Test Mame: A000090 Clock Resolution (variance) = 1.00000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78661.0000 1.0000 Iteration 23661.0000 1.0000 Iteration 23661.0000 1.0000 Iteration 23661.0000 1.0000 Iteration 237660.0000 1.0000 Iteration 512 78680.0000 Teate Rame: 10000 11.0000 Teate Rame: 449.2 microseconds. Iteration 2462.2 microseconds. Time to perform standard boolean operations on arrays of booleans. Por this test the arrays are BOT PACKED with the prayma 'PACK.' For this test the operations are performed on components in a loop. 1 Iteration 1 73400.0000 0.0000 Iteration 2 0.0000 0.0000 Iteration 3 73401.0000 0.0000 Iteration 12 73402.0000 0.0000 Iteration 12 73402.0000 0.0000 Iteration 12 73402.0000 0.0000 Iteration 12 73402.0000 0.0000 Iteration 12 1.0000
<pre>Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 8 40210.0000 40212.0000 0.0000 Iteration 1 40210.000 40212.0000 0.0000 Iteration 32 40215.0000 40215.0000 0.0000 Iteration 42 40210.0000 40245.0000 1.0000 Iteration 32 40276.0000 40245.0000 1.0000 Iteration 1024 40340.0000 40346.0000 6.0000 Test Mame: Tasking CPU Time: 571.1 microseconds Wall Time; 571.3 microseconds Test Mame; A000090 Clock resolution mesurement running Test Mame; A000091 Clock resolution using second differences of values returned by the function CPU_Time_Clock. Wmber of sample values is 7000 Clock Resolution (variance) = 0.0000000000000000000000000000000000</pre>	78655.0000 78655.0000 Iteration 64 78650.0000 78651.0000 Iteration 128 78660.0000 78651.0000 Iteration 238 78680.0000 78651.0000 Iteration 512 78680.0000 78651.0000 Iteration 512 78680.0000 78651.0000 Iteration 1024 78112.0000 78718.0000 Iteration 2000 Iteration 2000 Iteration 2000 Iteration 2000 Iteration 11.0000 Thest Baps: H00004 Class Name: Chaptar 13 CPU Time: 449.2 microseconds. Iteration Count: 7840.0000 78402.0000 0.0000 Iteration 1 78400.0000 0.0000 Iteration 2 78400.0000 78402.0000 78400.0000 78402.0000 0.0000 Iteration 12 78402.0000 1.0000 Iteration 256 78402.0000 1.0000 <t< td=""></t<>
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 6 40211.0000 40211.0000 0.0000 Iteration 18 40213.0000 40213.0000 0.0000 Iteration 32 40215.000 40220.0000 1.0000 Iteration 64 40218.0000 40220.0000 1.0000 Iteration 512 40228.0000 40228.0000 0.0000 Iteration 1024 40340.0000 40245.0000 1.0000 Iteration 1024 40340.0000 40245.0000 1.0000 Terest Name: Tosking CPU Time: 571.1 microseconds Mail Time: 571.3 microseconds Mail Time: 571.5 microseconds Mail Time: 57000 Clock resolution mesurement running Test Mame: A000090 Clock Resolution (average) = 1.00000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78661.0000 1.0000 Iteration 255 78660.0000 Teration 256 78660.0000 78680.0000 78661.0000 1.0000 Iteration 512 78680.0000 Teration 1024 6.0000 Iteration 1024 6.0000 Teration 2000 Iteration Teration 242. aicroseconds. Teration 2042. aicroseconds. Time to perform standard boolean operations on arrays of booleans. Pooleans. Por this test the arrays are BOT PACKED with the pragem 'PACK.' Por this test the operations are performed on components in a loop. Iteration 1 79400.0000 0.0000 1.0000 Iteration 2 79400.0000 0.0000 Iteration 32 79400.0000
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 8 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40212.0000 0.0000 Iteration 32 40215.0000 40220.0000 1.0000 Iteration 228.0000 0.0000 Iteration 228.0000 0.0000 Iteration 3245.0000 1.0000 Iteration 326.0000 40245.0000 1.0000 Iteration 310245.0000 1.0000 Iteration 310245.0000 1.0000 Iteration 310245.0000 40245.0000 Iteration 310245.0000 40245.0000 Iteration 310245.0000 40245.0000 Iteration 310245.0000 40245.0000 Iteration 311.3 microseconds Wall Time, 571.3 microseconds. Iteration Count: 1024 Test Mame: T000004 Class Mame: Tasking CPU Time; 571.3 microseconds. Iteration Count: 1024 Test Mame: A000090 Clock resolution measurement running Test Mame: A000090 Clock resolution measurement running Test Mame: A000091 Clock Resolution (average) = 1.00000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78661.0000 1.0000 Iteration 255 78660.0000 Teration 256 78660.0000 Teration 252 78680.0000 Teration 252 78680.0000 Teration 1024 6.0000 Teration 1024 6.0000 Teration 2048 11.0000 Tat Mame: 440.20 microseconds. Tisst test teration count: 2048 Test test testions on arrays of booleans. For this test the arrays are MOT PACKED with the pragma 'PACK.' For this test the arrays are MOT PACKED with the pragma 'PACK.' For this test the arrays are performed on components in a loop. Iteration 1 79400.0000 0.0000 1.0000 Iteration 2 0.0000 1.0000 1.0000 Iteration 32 79400.0000 0.0000 1.0000 Iteration 16 79400.0000 79400.0000 1.0000 Iteration 32 79400.0000 79400.0000 79400.0000 Tys
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40211.0000 0.0000 Iteration 8 40211.0000 40211.0000 0.0000 Iteration 8 40213.0000 40212.0000 0.0000 Iteration 32 40215.0000 40212.0000 0.0000 Iteration 32 40215.0000 40228.0000 0.0000 Iteration 228 40228.0000 40245.0000 1.0000 Iteration 328 40228.0000 40245.0000 1.0000 Iteration 328 40228.0000 40245.0000 1.0000 Iteration 312 40228.0000 40245.0000 1.0000 Iteration 312 40228.0000 40245.0000 1.0000 Iteration 313 40228.0000 40245.0000 5.0000 Iteration 312 40236.0000 40245.0000 Iteration 312 40236.0000 40245.0000 Iteration 312 40236.0000 40245.0000 Iteration 312 40236.0000 40245.0000 Iteration 513. sicroseconds. Iteration Count: 1024 Test Mame: 571.3 sicroseconds. Iteration Count: 1024 Test Mame: 571.3 sicroseconds. Iteration Count: 1024 Test Mame: A000090 Clock resolution measurement running Test Mame: A000090 Clock Resolution (average) = 1.000000000000000 seconds. Clock Resolution (average) = 1.00000000000000 seconds. Clock Resolution (average) = 1.00000000000000 seconds. Clock Resolution (average) = 0.0000000000000 seconds. Clock Resolution (average) = 0.00000000000000 seconds. Clock Resolution (average) = 0.0000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78661.0000 1.0000 Iteration 256 78660.0000 78680.0000 7869.0000 1.0000 Iteration 512 78680.0000 78680.0000 7869.0000 1.0000 Iteration 1000 1.0000 Teat 10000 5.0000 Iteration 10000 6.0000 Teat 10000 7810.0000 Test Name: Class Nall Time: 443.2 Sicroseconds Iteration Count: 2048 Test Description: Time to perform standard boolean operations on arrays of booleans. For this test the operations are performed on components in a loop. 1 1teration 1 7400.0000 0.0000 Iteration 2 0.0000 1 1teration 32 0.0000 1 78400.0000 0.0000 0.0000 Iteration 32 0.0000 1 78400.0000 0.0000 1 <t< td=""></t<>
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 4 40211.0000 40211.0000 0.0000 Iteration 16 40213.0000 40212.0000 0.0000 Iteration 32 40215.0000 40212.0000 0.0000 Iteration 32 40215.0000 40228.0000 0.0000 Iteration 228 40228.0000 4028.0000 0.0000 Iteration 228 40228.0000 4028.0000 0.0000 Iteration 322 40228.0000 4028.0000 0.0000 Iteration 324 40238.0000 40245.0000 1.0000 Iteration 324 40238.0000 40245.0000 5.0000 Iteration 1024 40340.0000 40246.0000 6.0000 Test Mame: T000004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: T000004 Class Mame: Tasking CPU Time: 571.3 microseconds. Iteration Count: 1024 Test Mame: A000090 Clock resolution mensurement running Test Mame: A000090 Clock resolution sensurement running Test Mame: A000090 Clock Resolution (average) = 1.00000000000000 seconds. Clock Resolution (average) = 1.0000000000000 seconds. Clock Resolution (average) = 0.0000000000000 seconds. Clock Resolution (average) = 0.0000000000000 seconds. Clock Resolution (average) = 0.0000000000000 seconds. Clock Resolution (average) = 0.00000000000000 seconds. Clock Resolution (average) = 0.0000000000000000000000000000000000	78655.0000 78655.0000 Iteration 64 78655.0000 78650.0000 Iteration 128 78660.0000 78650.0000 Iteration 120 78680.0000 7869.0000 Iteration 122 78680.0000 7869.0000 Iteration 1024 78112.0000 78718.0000 Iteration 1024 78770.0000 78718.0000 Iteration 2048 78770.0000 78718.0000 Iteration 2048 7870.0000 7871.0000 Test Rame: H00004 Class Name: Chapter 13 CPU Time: 449.2 Sicroseconds Iteration Count: Wall Time: 449.2 sicroseconds Test Description: Time to perform standard boolean operations on arrays of booleans. For this test the operations are performed on components in a loop. 1 79400.000 0.0000 Iteration 8 79401.0000 0.0000 Iteration 128 0.0000 1
Iteration 1 40210.0000 40210.0000 0.0000 Iteration 2 40210.0000 40210.0000 0.0000 Iteration 8 40211.0000 40211.0000 0.0000 Iteration 8 40213.000 40215.0000 0.0000 Iteration 32 40215.000 40215.0000 0.0000 Iteration 256 4024.0000 40215.0000 0.0000 Iteration 256 4024.0000 40228.0000 1.0000 Iteration 256 4024.0000 40245.0000 1.0000 Iteration 256 40244.0000 40245.0000 1.0000 Iteration 128 40340.0000 40346.0000 6.0000 Teet base: T00004 Class Name: Tasking CPU Time: 571.3 microseconds Wall Time: 571.3 microseconds Wall Time: 571.3 microseconds. Iteration Count: 1024 Teet Description: Test Base: T000004 Class Name: Tasking CDU Time: 571.3 microseconds. Iteration Count: 1024 Teet Description: Task entry call and return time measured One tasks ective: two entries, tasks in a package using select statement Test Mame: A000090 Clock resolution measurement running Toch resolution = 1.00000000000000000 seconds. Clock Resolution = 1.000000000000000000 seconds. Clock Resolution (versage) = 1.000000000000000 seconds. Clock Resolution (versage) = 0.00000000000000 seconds. Clock Resolution (versage) = 0.00000000000000 seconds. Clock Resolution (versage) = 0.0000000000000000 seconds. Clock Resolution (versage) = 0.000000000000000000000 seconds. Clock Resolution (versage) = 0.0000000000000000000000000000000000	78655.0000 78655.0000 0.0000 Iteration 64 78657.0000 78660.0000 78661.0000 1.0000 Iteration 236 78660.0000 Teration 236 1.0000 Iteration 312 78680.0000 Teration 312 78680.0000 Teration 312 0.0000 Teration 10000 1.0000 Teration 11.0000 1.0000 Teration 122 sicroseconds Feat Rame: H00004 Class Name: Chapter 13 CPU Time: 449.2 microseconds Iteration Count: 2048 Time to perform standard boolean operations on arrays of booleans. Por this test the arrays are MOT PACKED with the prayma 'PACK.' Por this test the operations are performed on components in a loop. 1 Iteration 1 73400.0000 79400.0000 0.0000 Iteration 2 73401.0000 79401.0000 0.0000 Iteration 6 73402.0000 79401.0000 0.0000 Iteration 128 73402.0000 79404.0000 1.0000 Teration 512 73402.0000 79404.0000 1.0000<

11(0	
Territor 1	Iteration 1 82835 0000 82935 0000 0 0000	
79770.0000 79770.0000 0.0000	Iteration 2 82936.0000 82936.0000 0.0000	
79770.0000 79770.0000 0.0000 Iteration 4	Iteration 4 82936.0000 82936.0000 0.0000	
79770.0000 79771.0000 1.0000 Iteration #	Iteration 8 82937.0000 82937.0000 0.0000	
/9//1.0000 /9//1.0000 0.0000 Iteration 16 79771 0000 70771 0000 0.0000	Iteration 16 82938.0000 82938.0000 0.0000	
79772.0000 79772.0000 0.0000 Teration 32 79772.0000 79772.0008 0.0000	110701100 32 82940.0000 82940.0000 0.0000 Iteration 64	
Iteration 64 79773.0000 79773.0000 0.0000	82944.0000 82944.0000 0.0000 Iteration 128	
Iteration 128 79774.0000_79775.0000 1.0000	82951.0000 82952.0000 1.0000 Iteration 256	
Iteration 256 79776.0000 79778.0000 2.0000	82965.0000 82967.0000 2.0000 Iteration 512	
Terretion 512 79780.0000 79783.0000 3.0000 Terretion 1024	82594.0000 82995.0000 2.0000 Iteration 1024 81051.0000 83056.0000 5.0000	
79788.0000 79794.0000 6.0000	Test Name: (000003	Class Name: Tasking
79803.0000 79816.0000 13.0000 Iteration 4096	CPU Time: 996.1 microseconds Wall Time: 996.1 microseconds.	Iteration Count: 1024
79833.0000 79860.0000 27.0000 Iteration 8192	Test Description: Task create and terminate measurement	
79894.0000 79947.0000 53.0000 Iteration 16384	with one task, no entries, when task is i using a task type in a package, no select	n a procedure statement, no loop,
	Thoration 1	
CPU Time: 17.7 microseconds Iteration Count: 16384	83167.0000 83167.0000 0.0000 Theration 2	
Test Description: Procedure call and return time measurement	83167.0000 83167.0000 0.0000 Iteration 4	
Compare to P000005 10 parameters, in INTEGER	\$3168.0000 \$3168.0000 0.0000 Iteration 8	
	\$3168.0000 \$3168.0000 0.0000 Iteration 16	
Iteration 1 80257.0000 80257.0000 0.0000	83169.0000 \$3170.0000 1.0000 Iteration 32	
100781100 2 80258.0000 80258.0000 0.0000 Terreton A	Iteration 64 83172.0000 #3172.0000 0.0000	
80258.0000 80258.0000 0.0000 Iteration 8	Iteration 128 83184.0000 83185.0000 1.0000	
80258.0000 80258.0000 0.0000 Iteration 16	Iteration 256 83199.0000 83200.0000 1.0000	
80259.0000 80259.0000 0.0000 Iteration 32	Iteration 512 83229.0000 83232.0000 3.0000	
80260.0000 80260.0000 0.0000 Iteration 64 00260 0000 0000 0.0000	Iteration 1024 83289.0000 83294.0000 5.0000	
BU262.0000 BU262.0000 0.0000 Iteration 128 BU266 DOOG BU267 DOOG 1.0000	Test Name: C000002 CPH Time: 1064.5 microseconds	Class Name: Tasking
Iteration 256 #0273.0000 #0275.0000 2.0000	Wall Time: 1064.5 microseconds. Test Description:	Iteration Count: 1024
Iteration 512 80288.0000 80290.0000 2.0000	Task create and terminate time measuremen with one task, no entries when task is in	a procedure,
Iteration 1024 80317.000080322.0000 5.0000	task defined and used in procedure, no se	elect statement, no loop
Iteration 2048 80374.0000 80385.0000 11.0000	Iteration 1	
Test Name: T000001 Class Name: Tasking	Iteration 2 83774.0000 83774.0000 0.0000	
Wall Time: 449.2 microseconds. Iteration Count: 2049	Iteration 4 \$3775.0000 \$3775.0000 0.0000	
Minimum rendervous, entry call and return time 1 task 1 entry , task inside procedure	Iteration 8 83775.0000 83775.0000 0.0000	
no select	Iteration 16 83776.0000 83776.0000 0.0000	
Iteration 1 80491 0000 80493 0000 0 0000	110701 32 83777.0000 83778.0000 1.0000	
Iteration 2 80491.0000 80491.0000 0.0000	83779.0000 83780.0000 1.0000 Iteration 128	
Iteration 4 80491.0000 80491.0000 0.0000	83783.0000 83784.0000 1.0000 Iteration 256	
Iteration # #0492.0000 80492.0000 0.0000	83791.0000 83792.0000 1.0000 Iteration 512	
Iteration 16 80494.0000 80494.0000 0.0000	\$3805.0000 \$3908.0000 3.0000 Iteration 1024	
10496.0000 80496.0000 0.0000	Iteration 2048 R1892.0000 #3904.0000 12.0000	
10000 80501.0000 1.0000 Iteration 128	Test Name: H000004	Class Name: Chapter 13
\$050\$.0000 80509.0000 1.0000 Iteration 256	CPU Time: 449.2 microseconds Wall Time: 449.2 microseconds.	Iteration Count: 2048
\$0524.0000 \$0525.0000 1.0000 Iteration 512	Test Description: Time to perform standard boolean operation	ns on arrays of booleans.
80555.0000 80558.0000 3.0000 Iteration 1024 	For this test the arrays are NOT PACKED w For this test the operations are performe	rith the pragma 'PACK.' d on components in a loop.
Tent Kane: T080004 Class Name: Tasking	Iteration 1	
CPU Time: 561.5 microseconds Wall Time: 561.5 microseconds. Iteration Count: 1024	84523.0000 84523.0000 0.0000 Iteration 2	
Test Description: Task entry call and return time measured	84523.0000 84523.0000 0.0000 Iteration 4	
One tasks active, two entries, tasks in a package using select statement	\$4523.0000 \$4523.0000 0.0000 Iteration 8	
T	34524.0000 84524.0000 0.0000 Iteration 16 84524.0000 84524.0000 0.0000	
Test Memorian inclusion	Iteration 32 #4524.0000 #4524.0000 0.0000	
Determine clock resolution using second differences of values returned by the function CPU Time Clock.	Iteration 64 84525.0000 84525.0000 0.0000	
Number of sample values is 7000	Iteration 128 84526.0000 84526.0000 0.0000	
Clock Resolution = 1.000000000000 seconds. Clock Resolution (average) = 1.000000000000 seconds.	Iteration 256 #4527.0000 #4529.0000 2.0000	
Clock Resolution (variance) = 0.0000000000000 seconds.	Iteration 512 04531.0000 84533.0000 2.0000 Therefine 1024	
Test Name: A000091 Class Name: Composite 1.1990 is time in millingrounds for one Devetors	10012100 1024 84536.0000 84542.0000 6.0000 Iteration 2048	
Test Description: Reinhold P. Weicker's DHRYSTOWE composite benchmark	84548.0000 84559.0000 11.0000 Iteration 4096	
	84571.0000 84592.0000 21.0000 Iteration 8192	
Tast Hama: A000092 Class Hame: composite	\$4617.0000 \$4659.0000 42.0000 Iteration 16384	
Average time per cycle : 2824.67 milliseconds	\$4708.0000 \$4793.0000 \$5.0000 ***** Incomplete Measurement *****	
Average whethiche fating i 334 KWIPS	Test Name: P000001 CPU Time: 7.9 microseconds	Class Name: Procedure

Mall Time: 7.9 microseconds. Tteration Count: 16384 Test Description: Procedure call and return time (may be zero if automatic inlining) procedure is local no parameters

Iteration 1 142.0000 142.0000 0.0000 Iteration 2 142.0000 142.0000 0.0000 Iteration 4 143.0000 143.0000 0.0000 Iteration 16 144.0000 140.0000 0.0000 Iteration 16 145.0000 147.0000 0.0000 Iteration 1251.0000 Iteration 256 150.0000 151.0000 Iteration 256 150.0000 153.0000 Iteration 512 201.0000 263.0000 Iteration 1024 257.0000 263.0000 Iteration 1024 257.0000 263.0000 Iteration 1024 257.0000 145.0000 Iteration 1024 201.0000 263.0000 Iteration 1024 Iterati Iteration 1 \$4\$93.0000 84893.0000 0.0000 Iteration 2 84893.0000 84893.0000 0.0000 Iteration 4 84893.0000 \$4893.0000 0.0000 teration 8 44893.0000 84894.0000 Iteration 16 84894.0000 84894.0000 Iteration 32 84894.0000 84894.0000 1.0000 0.0000 0.0000 Iteration 64 84895.0000 84896.0000 1.0000 84895.0000 84896.0000 Iteration 128 84897.0000 84897.0000 Iteration 256 84899.0000 84900.0000 Iteration 512 84903.0000 84906.0000 0.0000 1.0000 3.0000
 #4603.0000
 #4905.0000
 3.0000

 tteration
 1024
 6.0000

 tteration
 2048
 17.0000
 6.0000

 tteration
 2048
 13.0000
 13.0000

 tteration
 2048
 27.0000
 13.0000

 tteration
 695
 27.0000
 13.0000

 tteration
 6192
 27.0000
 1000
 53.0000

 tteration
 1518
 53.0000
 106.0000
 106.0000
 Test Hame: C000001 Class Hame: Taski CPU Time: 996.1 microseconds Iteration Count: Test Description: Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select statement, no loop, Class Name: Tasking Iteration Count: 1024 Iteration 1 373.0000 373.0000 373.0000 373.0000 10000 10000 374.0000 374.0000 374.0000 374.0000 374.0000 374.0000 374.0000 376.0000 376.0000 376.0000 378.00000 378.00000 378.0000 378.0000 378.00000000 378 Test Mame: P000010 CPU Time: 17.7 microseconds Wall Time: 17.7 microseconds. Test Description: Procedure call and return time measurement Compare to P000005 10 parameters, in INTEGER Class Name: Procedure 0.0000 Iteration Count: 16384 0.0000 0.0000 0.0000 0.0000 Iteration 1 85380.0000 85380.0000 0.0000 1538.0000 25380.0000 15eration 2 85380.0000 85380.0000 15eration 4 85381.0000 85381.0000 15eration 8 85381.0000 85381.0000 0.0000
 10000
 10000
 10000

 10010
 382.0000
 1.0000

 10010
 382.0000
 1.0000

 10010
 382.0000
 1.0000

 10010
 380.0000
 1.0000

 10010
 390.0000
 1.0000

 10010
 1000
 1.0000

 10010
 405.0000
 1.0000

 10010
 417.0000
 3.0000

 10010
 493.0000
 499.0000
 0.0000 0.0000 0.0000 Iteration 16 \$5382.0000 \$5382.0000
 1teration
 16

 83382.0000
 85382.0000
 0.0000

 1teration
 32
 0.0000

 1teration
 32
 0.0000

 1teration
 64
 0.0000

 1teration
 64
 0.0000

 1teration
 64
 0.0000

 1teration
 12
 0.0000

 1teration
 12
 0.0000

 1teration
 12
 0.0000

 1teration
 12
 0.0000

 1teration
 1000
 0.0000

 1teration
 518
 0.0000

 1teration
 1000
 0.0000

 1teration
 518
 0.0000
 1.0000

 5497.0000
 5508.0000
 11.0000
 0.0000 Test Hame: C000002 CPU Time: 1035.2 Class News: Tasking 1035.2 microseconds 1035.2 microseconds. CPU Time: 1035.2 microseconds. Iteration Count: 102 Test Description: Task create and terminate time measurement. with one task, no entries when task is in a procedure, task defined and used in procedure, no select statement, no loop Iterstion Count: 1024 Iteration 1 978.0000 978.0000 Iteration 2 978.0000 978.0000 Iteration 4 979.0000 979.0000 Iteration 5 979.0000 979.0000 Iteration 16 980.0000 980.0000 Iteration 64 983.0000 980.0000 Iteration 256 995.0000 995.0000 Iteration 512 1009.0000 995.0000 Iteration 226 995.0000 995.0000 Iteration 2024 1038.0000 1044.0000 Iteration 2044.0000 Iteration 2044.0000 Iteration 2044.0000 Iteration 2044.0000 Iteration 2044.0000 1096.0000 1108.0000 Test Mage: T000001 11.0000 Test Mage: T000001 Clat CPU Time: 454.1 microseconds Iter Test Description: Minissa rendervous, entry call and return time 1 task i entry, task inside procedure no select 0.0000 Class Home: Tasking 0.0000 Iteration Count: 2048 0.0000 0.0000 0.0000 1.0000 Iteration 1 #5614.0000 #5614.0000 0.0000 1.0000 Iteration 2 \$5615.0000 \$5615.0000 0.0000
 #5615.0000
 #5615.0000

 Iteration
 #

 #5615.0000
 #5615.0000

 Iteration
 #

 #5616.0000
 #5616.0000

 Iteration
 #

 #5617.0000
 #5617.0000
 1.0000 0.0000 1.0000 0.0000 3.0000 0.0000 6.0000 Iteration 32 85620.0000 85620.0000 0.0000
 #5520.0000
 #5520.0000
 0.0000

 Iteration
 64
 0.0000

 #5624.0000
 85624.0000
 0.0000

 Iteration
 128
 0.0000

 #5632.0000
 85633.0000
 1.0000

 Iteration
 256
 1.0000

 Iteration
 756
 1.0000

 Iteration
 756
 1.0000

 Iteration
 128
 1.0000

 Iteration
 126
 1.0000

 Iteration
 123
 3.0000

 Iteration
 1024
 3.0000

 Iteration
 1024
 3.0000
 12.0000 Test Name: H000004 Class Name: Chapter 13 CPU Time: 444.3 microseconds. Iteration Count: 2048 Wall Time: 444.3 microseconds. Iteration Count: 2048 Test Description: Time to perform standard boolean operations on arrays of booleans. For this test the arrays are NOT PACERD with the progma 'PACK.' For this test the operations are performed on components in a loop. Test Name: T000004 Class N CPU Time: 556.6 microseconds Wall Time: 556.6 microseconds. Iterati Test Description: Task entry call and return time measured One tasks active. two entries, tasks in a package using select statement Iteration 1 1798.0000 1798.0000 Iteration 2 1798.0000 1798.0000 Iteration 4 1799.0000 1798.0000 Class Name: Tasking 0.0000 Iteration Count: 1024 0.0000 0.0000 1798.0000 1798.0000 Iteration 8 1798.0000 1798.0000 Iteration 16 1799.0000 1799.0000 Iteration 32 1799.0000 1799.0000 0.0000 Test Name: A000090 Clock resolution measurement running Test Description: Determine clock resolution using second difference of values returned by the function CPU_Time_Clock. 0.0000 0.0000 Iteration (1800.0000 64 10 1800.0000 0.0000 Iteration 128 1801.0000 1892.0000 Rumber of sample values is 7000 Clock Resolution = 1. Clock Resolution (verage) = 1. Clock Resolution (variance) = 0. 1.0000 1801.0000 1902.0000 Iteration 256 1803.0000 1804.0006 Iteration 512 1806.0000 1809.0000 Iteration 1024 1812.0000 1818.0000 1.000000000000000 seconds. 1.00000000000000 seconds. 0.00000000000000 seconds. 1.0000 3.0000 Iteration 1024 1012.0000 1018.0000 Iteration 2048 1025.0000 1037.0000 Iteration 4096 1050.0000 1074.0000 Iteration 9192 Test Name: A000091 Class Name: Composite 1.2000 is time in millimeconds for one Dhrystone Test Description: Reinhold P. Weicher's DNRYSTOWE composite benchmark 6 0000 12.0000 24.0000 48.0000 Test Hames A000092 Class Name: composite

Average time per cycle : -342758.67 millimeconds Average Whetstone rating : 281 KWIP5

Iteration 16384 1998.0000 2093.0000 95.0000 Test Hame: P000001 Class Hame: Procedure CPU Time: 4.3 microssconds Iteration Count: 16384 Test Description: Procedure call and return time (may be zero if automatic inlining) procedure is local no parameters
 NO parameters

 Iteration 1 2197,0000 2197.0000 0.0000

 Iteration 2 198,0000 2198.0000 0.0000

 Iteration 4 2198.0000 2198.0000 0.0000

 Iteration 5 2198.0000 2198.0000 0.0000

 Iteration 12 2198.0000 2199.0000 1.0000

 Iteration 32 2199.0000 2199.0000 0.0000

 Iteration 12 2199.0000 2199.0000 0.0000

 Iteration 32 2199.0000 2290.0000 1.0000

 Iteration 128 2200.0000 2200.0000 1.0000

 Iteration 128 2200.0000 2244.0000 1.0000

 Iteration 1024 2207.0000 2210.0000 7.0000

 Iteration 1024 2210.0000 2210.0000 1.0000

 Iteration 1024 2210.0000 221.0000 1.0000

 Iteration 1024 2210.0000 221.0000

 Iteration 1024 2210.0000 221.0000

 Iteration 1024 223.0000 225.0000

 Iteration 8192 2310.0000 235.0000

 Iteration 8192 2310.0000 235.0000

 Iteration 8192 2310.0000 235.0000

 Iteration 8192 2310.0000 235.0000

 Iteration 16184 2436.0000

 Iteration 16184 2436.0000
 Test Hame: P000010 CPU Time: 18.3 microseconds Wall Time: 18.3 microseconds. Test Description: Procedure call and return time measurement Compare to P000005 10 parameters, in LWTEGER Class Name: Procedure Iteration Count: 16384 Iteration 1 2874.0000 2674.0000 Iteration 2 2475.0000 2675.0000 Iteration 4 2475.0000 2675.0000 Iteration 9 2475.0000 2675.0000 Iteration 12 2477.0000 2677.0000 Iteration 12 2477.0000 2677.0000 Iteration 128 2477.0000 2677.0000 Iteration 128 2487.0000 2674.0000 Iteration 128 2480.0000 2482.0000 Iteration 112 2705.0000 2795.0000 Iteration 1024 2791.0000 2802.0000 Iteration 2008 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 2.0000 1.0000 5.0000 Test Mass: T00001 11.0000 Test Mass: T000001 Cla CPU Time: 454.1 microseconds Wall Time: 454.1 microseconds. Ite: Test Description: Winimum renderwoos, entry call and return time 1 task 1 entry, task inside procedure no select 11.0000 Class Name: Tasking Iteration Count: 2048 Iteration 1 2908.0000 2008.0000 Iteration 2 2908.0000 2008.0000 Iteration 4 2909.0000 2009.0000 Iteration 8 2910.0000 2910.0000 Iteration 16 2911.0000 2911.0000 Iteration 32 2913.0000 2918.0000 Iteration 124 2937.0000 2918.0000 Iteration 256 2942.0000 2943.0000 Iteration 124 2937.0000 2918.0000 Iteration 124 2937.0000 2917.0000 Iteration 1024 3938.0000 1044.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 1.0000 1,0000 3.0000 6.0080 Test Mamer 17000004 Class I CPU Time: 556.4 microseconds Wall Time: 556.4 microseconds. Iterat Test Description: Task entry call and return time measured One tasks active, two entries, tasks in a package using select statement Class Name: Tasking Iteration Count: 1024

Bibliography

- 1. Boehm, B.W., A Spiral Model of Software Development and Enhancement, in Software Engineering Project Management, R.H. Thayer, Editor. 1987, IEEE Computer Society Press: Los Alamitos, CA. p. 128 - 142.
- 2. Yeh, R.T., et al., A Commonsense Management Model, in IEEE Software. 1991, p. 23 - 33.
- 3. Pyster, A., The Synthesis Process for Software Development, in System and Software Requirements Engineering, R.H. Thayer and M. Dorfman, Editor. 1990, IEEE Computer Society Press: Los Alamitos, CA. p. 528 -538.
- 4. Shaw, A.C., Reasoning About Time in Higher-Level Language Software. IEEE Transactions on Software Engineering, 1989. 15(7): p. 875 - 889.
- 5. Park, C.Y. and A.C. Shaw. A Source-Level Tool for Predicting Deterministic Execution Times of Programs. Department of Computer Science, University of Washington, (Technical Report 89-09-12). September 13, 1989.
- 6. Shaw, A.C. Towards a Timing Semantics For Programming Languages. in Third Annual Workshop, Foundations of Real-Time Computing. 1990. Washington, DC: Office of Naval Research.
- Goos, G., W.A. Wulf, A. Evans Jr., and K.J. Butler, ed. DIANA: An Intermediate Language for Ada. Lecture Notes in Computer Science, ed. G. Goos and J. Hartmanis. 1983, Springer-Verlag: Berlin. 201 pages.
- 8. Shaw, A.C. Communiating Real-Time State Machines. Department of Computer Science and Engineering, University of Washington, (Technical Report 91-08-09). August 1991.
- 9. Haase, V.H., *Real-Time Behavior of Programs*. IEEE Transactions on Software Engineering, 1981. **7**(5): p. 494 501.
- 10. Halang, W.A. A Priori Execution Time Analysis for Parallel Processes. in Proceedings of the Euromicro Workshop on Real-Time. 1989. IEEE Computer Society Press.

- 11. Mok, A.K., P. Amerasinghe, M. Chen, and K. Tantisirivat. Evaluating Tight Execution Time Bounds of Programs by Annotations. in 6th IEEE Workshop on Real-Time Operating Systems and Software. 1989. Pittsburgh:
- Puschner, P. and C. Koza, Calculating the Maximum Execution Time of Real-Time Programs. The Journal of Real-Time Systems, 1989. 1(2): p. 159 - 176.
- 13. Kenny, K.B. and K.-J. Lin, Measuring and Analyzing Real-Time Performance, in IEEE Software. 1991, p. 41 49.
- 14. Glicker, S.M. and F.A. Hosch. Toward Automating the Execution Timing Analysis of Ada Tasks in Event-Driven Real-Time Systems. Applied Research Laboratories, The University of Texas at Austin, (Technical Report ARL-TR-91-4). 19 February 1991.
- Knuth, D.E., The Art of Computer Programming. Second ed. Addison-Wesley Series in Computer Science and Information Processing, ed. R.S. Varga and M.A. Harrison. Vol. 1/Fundamer: al Algorithms. 1973, Reading, Massachusetts: Addison-Wesley Publishing Company. 634 pages.
- 16. Hoare, C.A.R., An Axiomatic Basis for Computer Programming. Communications of the ACM, 1969. 12(10): p. 576-580.
- 17. Shaw, M. A formal System for Specifying and Verifying Program Performance. Department of Computer Science, Carnegie-Mellon University, (Technical Report CMU-CS-79-129). 21 June 1979.
- 18. Walden, E. and C.V. Ravishankar, A Survey of Hard Real-Time Scheduling Algorithms. unpublished draft, 1990. .
- 19. Cornhill, D., et al. Limitations of Ada for Real-Time Scheduling. in Proceedings of the International Workshop of Real-Time Ada Issues. 1987. Moretonhampstead, Devon, UK: ACM SIGAda.
- 20. Baker, T.P. and A. Shaw. The Cyclic Executive Model and Ada. in The Real-Time Systems Symposium. 1988. Huntsville, AL: IEEE Computer Society.
- 21. Grogono, P., Comments, Assertions, and Pragmas. SIGPLAN Notices, 1989. 24(3): p. 79 84.
- 22. Lin, K.-J. and J.W.S. Liu. FLEX: A Language for Real-Time Systems Programming. in Third Annual Workshop, Foundations of Real-Time Computing. 1990. Washington, DC: Office of Naval Research.
- 23. Military Standard Ada Programming Language, ANSI/MIL-STD-1815A, U.S. Department of Defense, Ada Joint Program Office, (January 1983).

- 24. Motorola, MC68030 Enhanced 32-Bit Microprocessor User's Manual. Third ed. 1990, Englewood Cliffs, New Jersey: Prentice-Hall, Inc.
- 25. Shin, K.G. HARTS: A Distributed Real-Time Architecture. in Third Annual Workshop, Foundations of Real-Time Computing. 1990. Washington, DC: Office of Naval Research.
- 26. Stankovic, J.A. and K. Ramamritham, The Spring Kernel: A New Paradigm for Real-Time Operating Systems. SIGOPS, 1989. 23(3): p. 54 71.
- 27. Chen, M.-S., K.G. Shin, and D.D. Kandlur, Addressing, Routing, and Broadcasting in Hexagonal Mesh Multiprocessors. IEEE Transactions on Computers 1990. **39**(1): p. 10 - 18.
- 28. Hoare, C.A.R., Communicating Sequential Processes. Prentice-Hall International Series in Computer Science, ed. C.A.R. Hoare. 1985, Englewood Cliffs, New Jersey: Prentice-Hall. 256 pages.
- 29. Hoare, C.A.R., Communicating Sequential Processes. Communications of the ACM, 1978. 21(8): p. 666-677.
- 30. Park, C.Y. and A.C. Shaw. Experiments With a Program Timing Tool Based on Source-Level Timing Schema. in IEEE Real-Time Systems Symposium. 1990. Lake Buena Vista, Florida: IEEE Computer Society Press.
- 31. Woodbury, M.H., Workload Characterization of Real-Time Computing Systems. 1988, The University of Michigan:
- 32. Chu, W.W., C.-M. Sit, and K.K. Leung, Task Response Time For Real-Time Distributed Systems With Resource Contentions. IEEE Transactions on Software Engineering, 1991. 17(10): p. 1076 - 1092.
- 33. Gerber, R. and I. Lee. Communicating Shared Resources: A Model for Distributed Real-Time Systems. in IEEE Real-Time Systems Symposium. 1989. Santa Monica, CA: IEEE Computer Society Press.