

AD-A257 849



2

WL-TR-92-7015

Architecture/Environment Evaluation

Capt George York

**Wright Laboratory, Armament Directorate
Weapon Flight Mechanics Division
Guidance and Control Branch
Eglin AFB FL 32542-5000**

**DTIC
ELECTE
NOV 17 1992**
S C D

OCTOBER 1992

FINAL REPORT FOR PERIOD JUNE 1989 - FEBRUARY 1992

Approved for public release; distribution is unlimited.

425586 **92-29626**

5108

WRIGHT LABORATORY, ARMAMENT DIRECTORATE
Air Force Materiel Command ■ United States Air Force ■ Eglin Air Force Base

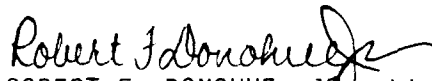
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise as in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This technical report has been reviewed and is approved for publication.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service (NTIS), where it will be available to the general public, including foreign nationals.

FOR THE COMMANDER



ROBERT F. DONOHUE, JR., Lt Col, USAF
Chief, Weapon Flight Mechanics Division

Even though this report may contain special release rights held by the controlling office, please do not request copies from the Wright Laboratory, Armament Directorate. If you qualify as a recipient, release approval will be obtained from the originating activity by DTIC. Address your request for additional copies to:

Defense Technical Information Center
Cameron Station
Alexandria VA 22304-6145

If your address has changed, if you wish to be removed from our mailing list, or if your organization no longer employs the addressee, please notify WL/MNAG, Eglin AFB FL 32542-5000, to help us maintain a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE October 1992	3. REPORT TYPE AND DATES COVERED Final June 1989 - February 1992
----------------------------------	--------------------------------	---

4. TITLE AND SUBTITLE Architecture/Environment Evaluation	5. FUNDING NUMBERS PE: 62602F PR: 2567 TA: 01 WU: 59
6. AUTHOR(S) Capt George York	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Wright Laboratory, Armament Directorate Weapon Flight Mechanics Division Guidance and Control Branch (WL/MNAG) Eglin AFB FL 32542-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
---	--

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING / MONITORING AGENCY REPORT NUMBER WL-TR-92-7015
---	---

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A
---	---------------------------------

13. ABSTRACT (Maximum 200 words)
The objective of the Architecture/Environment Evaluation project was to determine the ideal microprocessor and the Ada software environment for embedded guidance, navigation, and control processing for tactical missiles. Off-the-shelf microprocessors and compilers were evaluated using the Optimum Guidance Law Implementation (OGLI) benchmarks. The processors included the MIPS R3000, Sun SPARC, Motorola 88000, IBM System/6000, TI TMS320C30, INMOS transputer T800, Intel 80386, and the DEC MicroVAX II. This research also included an experiment implementing OGLI on a parallel computer (transputer) to improve performance.

14. SUBJECT TERMS Microprocessor Reduced Instruction Set Computer Missile Guidance Benchmark Ada Parallel Processing	15. NUMBER OF PAGES 51
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL
---	--	---	----------------------------------

PREFACE

This program was conducted by the Wright Laboratory, Armament Directorate, Eglin AFB FL 32542-5000. Capt George York of the Guidance Control Branch (WL/MNAG) managed this program. The program was conducted during the period from 15 June 1989 through 28 February 1992.

Commercial products mentioned in this report are sometimes identified by manufacturer or brand name. Performance statistics for these products are presented for specific benchmarks used in this Research and Development (R&D) effort. These results can not be generalized to assume one product is better than another. Given different benchmarks and/or different system designs, the relative performance between products could change considerably. The mentioning of these products and performance statistics is necessary for an understanding of the R&D effort but does not constitute endorsement of these items by the U.S. Government.

~~CONFIDENTIAL~~

Accession For	
NTIS CMAS	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist and/or	
Dist	Special
A-1	

TABLE OF CONTENTS

Section	Title	Page
I	INTRODUCTION	1
	1. Objective	1
	2. Background	2
	a. Tactical Missile Requirements	2
	b. RISC versus CISC Debate	2
	c. Previous Projects - GISA	3
	d. Tasks for this Project	4
II	GUIDANCE, NAVIGATION, AND CONTROL (GN&C) BENCHMARKING	6
	1. Philosophy of Benchmarking	6
	a. All Levels of System Impact Performance	6
	(1) Microprocessor	6
	(2) Computer Architecture	8
	(3) Operating System/Run-Time	8
	(4) Assembler/Linker	9
	(5) Ada Compiler	9
	(6) Algorithm/Application	11
	b. Real Requirement	11
	c. Measurements	11
	2. Benchmarks for GN&C Application	12
	a. CAMP	13
	b. ACEC	13
	c. PIWG	13
	d. OGLI	14
	3. Benchmarking Tools	14
	4. Candidate Microprocessors	15
	a. DEC MicroVax II (CISC)	15
	b. Intel 80386 (CISC)	16
	c. Motorola 68040 (CISC)	17
	d. MIPS R3000 (RISC)	17
	e. Sun SPARC (RISC)	18
	f. Motorola 88000 (RISC)	18
	g. IBM System 6000 (RISC)	19
	h. Intel i960 (RISC)	20
	i. Texas Instruments (TI) TMS320C30 (Signal)	20
	j. INMOS Transputer T800 (Parallel)	20
	5. Results and Analysis	21
	a. Execution Speed (VAX-MIPS)	29
	b. Compile Time	30
	c. Code Expansion	31
III	PARALLEL IMPLEMENTATION OF OGLI	32
	1. Transputer Background	32
	2. Approach	32
	a. Partition	33
	b. Timing Analysis	33
	c. Throughput/Communication Analysis	33
	3. Results	34

TABLE OF CONTENTS (Concluded)

Section	Title	Page
IV	CONCLUSION AND RECOMMENDATIONS	38
	REFERENCES	39
	BIBLIOGRAPHY	40

LIST OF FIGURES

Figure	Title	Page
1	All Levels of System Impact Performance	7
2	Pipeline Example	10
3	Optimum Guidance Law Implementation	14
4	OGLI Benchmarks	24
5	OGLI Benchmarks at 20 MHz	25
6	OGLI Benchmarks at Max Clock	26
7	Step 1 Benchmark at 20 MHz	27
8	Code Expansion and Compile Time	28
9	Partition and Dependency Analysis	35
10	OGLI Processing Flow on Three Transputers	36

LIST OF TABLES

Table	Title	Page
1	OGLI Benchmark Results	22
2	Step 1 Benchmark Results	23
3	Code Expansion and Compile Times	23
4	OGLI Benchmarks on Transputers	37

LIST OF ABBREVIATIONS AND ACRONYMS

ACEC	Ada Compiler Evaluation Capability
CAMP	Common Ada Missile Package
CISC	Complex Instruction Set Computer
CMOS	Complimentary Metal Oxide Semiconductor
CPU	Central Processing Unit
CSALI	Core Set of Assembly Language Instructions
DARPA	Defense Advanced Research Projects Agency
ECL	Emitter Coupled Logic
FLOPS	Floating Point Operations Per Second
Gbyte	gigabyte
GISA	Guidance Instruction Set Architecture
GN&C	Guidance, Navigation, and Control
IMU	Inertial Measurement Unit
ISA	Instruction Set Architecture
kbyte	kilobyte
Mbyte	megabyte
MIPS	Million Instructions Per Second
MMU	Memory Management Unit
NOOPs	No Operations
OGLI	Optimal Guidance Law Implementation
PIWG	Performance Issues Working Group
RAM	Random Access Memory
RISC	Reduced Instruction Set Computers
ROM	Read Only Memory
SPARC	Scalable Processor Architecture
Tbyte	terabyte

SECTION I

INTRODUCTION

1. OBJECTIVE

The Architecture/Environment Evaluation project was initiated to determine the ideal microprocessor and Ada software environment for embedded Guidance, Navigation, and Control (GN&C) processing for tactical missiles. Our goal was to reduce cost by evaluating state-of-the-art, off-the-shelf technology that has been improving at an amazing rate due to the commercial market. Previously we had planned to fund the research for a specialized GN&C Very High Speed Integrated Circuit (VHSIC) chip. However, this VHSIC chip would have a limited market, high cost, and lag in performance due to the commercial markets greater efficiency over government procurement. Therefore, this research focuses on evaluating the current off-the-shelf technology. This report discusses the benchmarking of various microprocessor/compiler combinations with specific Ada GN&C benchmarks. Also included in this report is a discussion of an experiment using a parallel computer architecture to increase performance.

Specific project objectives included:

- a. Developing an Ada benchmark suite representative of GN&C software for future missile systems. This is a hybrid suite made up of the Common Ada Missile Packages (CAMP) benchmarks, the Ada Compiler Evaluation Capability (ACEC) benchmarks, the Performance Issue Working Group (PIWG) benchmarks, and modern control algorithms, which includes the OGLI benchmarks.
- b. Acquiring 32-bit microprocessor systems with accompanying Ada tool sets. These processors include reduced instruction set computers (RISC), complex instruction set computers (CISC), signal processors, and parallel processors.
- c. Benchmarking the performance of the candidate microprocessors in executing the GN&C algorithms, and evaluating the unique Instruction Set Architecture (ISA) features of the various processors that lead to increased performance. This thorough evaluation will determine whether a specific processor for guidance and control should be manufactured in VHSIC technology or whether an off-the-shelf microprocessor will be sufficient.
- d. Evaluating the Ada software environments with respect to code efficiency (speed and size), compilation time, user-friendliness, design support, debugging support, testing support, and reliability.
- e. Implementing GN&C algorithms in parallel for various parallel architectures and determine the performance gain that can be

obtained. Evaluate the Ada software development tools for parallel computers.

2. BACKGROUND

a. Tactical Missile Requirement

Virtually all future tactical weapons will make use of a computer, either ground based for development and support or embedded in the weapon itself. As the performance requirements placed on precision-guided munitions become more complex, their embedded guidance processors must be redesigned to satisfy these new demanding requirements. In order to meet increasingly challenging threats, the weapon guidance processor must be designed for quick and efficient response times. These processing requirements are exceeding that of the 16-bit MIL-STD-1750A processor, especially when executing in an Ada environment. Several new technologies offer great potential for constructing improved guidance processors. A notable near-term approach features using a streamlined computer ISA, based on RISC concepts, optimized for missile guidance and control applications.

b. RISC Versus CISC Debate

Over the past two decades, as chip technology improved the allowance of gates on a chip, microprocessor designers used this extra room to add more complex instructions, addressing schemes, and control techniques, thus attempting to get more work done with a single instruction. This led to the term CISC. Although assembly programmers liked these complex instructions, compilers rarely use a large number of these instructions and stick to a few basic instructions. These excess instructions add complexity in the circuitry, which results in slower instruction cycles and less room on the chip that could be used for something else. Each CISC instruction takes several clock cycles to execute, and the number of cycles per instruction can vary greatly resulting in complex timing requirements.

RISC technology focuses on simple instructions that can be executed in one machine cycle and on architectures where only load and store instructions can access memory. These features tend to make RISC processors execute much faster than CISC. The single cycle per instruction removes the need for complex timing circuitry. Only having load and store access to memory, instead of a large number of addressing modes, reduces the number of instructions and the timing problems due to complex memory accesses. The disadvantage is several RISC instructions are needed to do the equivalent work of one CISC instruction. However, due to the high speed at which the RISC instructions are executed and the fact that CISC instructions take several cycles to execute, RISC processors have consistently shown better processing performance. In order to process instructions quickly, RISC processors need quick cache memory and memory controllers to feed them, which is expensive. CISC designers are also starting to find the need for fast cache memory as they improve their processors to keep up with RISC. Experts hypothesize that CISC designs

will take on more RISC features, and RISC designs will add more traditional CISC features, eventually reaching an optimum medium.

Recently, some high powered signal processors and parallel processors have come on the market with Ada compilers. Signal processors are specifically designed for mathematically intense applications, such as matrix and image processing. Parallel processors have special communication links built on chip to minimize the overhead in communicating with other processors. Many design features overlap between signal, parallel, RISC and CISC.

c. Previous Projects - GISA

Two Guidance Instruction Set Architecture (GISA) programs were initiated to address these issues. The product of GISA-1 was an experimental ISA design optimized for GN&C systems in Ada. In GISA-2, the GISA-1 design was analyzed along with several off-the-shelf processors, to determine a candidate architecture on which to base the guidance computer architecture. Neither the GISA-1 design nor any existing architecture met all the guidance and control requirements; however, the Defense Advanced Research Projects Agency (DARPA) Core Set of Assembly Language Instructions (CSALI) was close. The CSALI architecture is a RISC design. Based on the analysis of missile guidance and control code, the CSALI was modified for optimum execution. This design was implemented in a hardware brassboard, built around the million instructions per second (MIPS) R3000 (a commercial application of CSALI). An Ada tool set was developed with an embedded run-time system.

In the second quarter of 1990, the evaluation of the GISA-2 processor versus a MIL-STD-1750A, a 68020, and a 80960 RISC processor in processing the CAMP benchmarks was completed. This evaluation demonstrated that the GISA 32-bit RISC architecture far out-performed the 16-bit MIL-STD-1750 and the 32-bit 68020 CISC. GISA out-performed its competitor, the 80960 RISC, and performed similarly to a MIPS R3000 system. Since the off-the-shelf R3000 system performed similarly, it would be more cost effective to use off-the-shelf commercial products than to manufacture the GISA architecture in VHSIC. However, a VHSIC chip would provide a small package with high throughput. A VHSIC chip would allow unique embedded features to be built on-chip, which are needed for real-time GN&C computers and not found in commercial microprocessors. Examples of these features include external real-time counters, interrupt logic, communication logic to buses or memory, or any of the external logic that interface between the chip and external devices.

The competition of industry in the RISC market has resulted in several highly capable microprocessors. In the next year, several companies will introduce new versions of their processors, doubling their present capability. If the GISA-2 design were made into a VHSIC processor in a GISA-3 program, by the time the three-year procurement cycle was over, the GISA-3 chip would be old and slow technology compared to the commercial

market. More information about the GISA-2 project can be found in the GISA final report from the Defense Technical Information Center (DTIC); see Reference 1.

d. Tasks for this Project

Due to these issues, this in-house architecture/environment evaluation was started, taking into account advances in industry, other government programs, and our GN&C requirements. There were five specific tasks for this project.

(1) Develop Benchmark Suite

An Ada software benchmark suite representative of GN&C requirements has been developed. The CAMP benchmarks, ACEC benchmarks, PIWG benchmarks and our in-house modern control algorithms (OGLI) make up the benchmark suite. OGLI has been divided into four main benchmarks representing the phases of flight: Full_Midcourse, Ramping_Midcourse, Transition, and Terminal. The process of executing the benchmarks has been automated and made portable to all Ada systems, following the ACEC benchmarking methodology and tool set.

(2) Select Candidate Processors and Compilers

Several candidate processors with accompanying Ada software environments have been acquired, including the RISC (GISA-2 or MIPS R3000, IBM System 6000, Motorola 88000, and Sun Scalable Processor Architecture known as SPARC), the CISC (Motorola 68020 and Intel 80286), the signal processor (Texas Instruments C30), and the parallel processor (Seven Inmos transputer T800s). Presently the candidates not represented include the upgraded CISC (Intel 80486 and Motorola 68040), RISC (MIPS R4000 and R5000, and the Intel 80960), signal processor (Texas Instruments C40), and the parallel processor (Inmos transputer H-1).

(3) Processor Evaluation

The GISA-2 processor has been evaluated versus the other RISC, CISC, signal, and parallel microprocessors in executing the OGLI benchmarks. The unique ISA design features of the various processors that lead to increased performance have been examined. An optimum ISA design will eventually be determined and the best candidate microprocessor chosen. This is a continual process, with the "ideal" candidate updated as reflected by changes in microprocessor technology.

(4) Compiler Evaluation

The efficiency with respect to execution speed, code expansion, and compile time of the Ada compilers have been measured using the same benchmark code. The Ada tool sets have been evaluated with respect to "user-friendliness," debugging support, testing support, design support, and code reliability.

(5) Parallel Processing

The OGLI GN&C algorithms were studied to determine their potential to be implemented in parallel. The algorithms were decomposed into independent blocks of code (Steps 1 through 25, which were further refined and are presented in Section III). These steps in turn were decomposed further into finer independent tasks, resulting in various levels of granularity. These steps were run independently on a single processor. The timing information for all the tasks were used to estimate the performance improvements due to implementing in parallel at different levels of granularity and for different numbers of processors. The transputer board with seven processors was easily configured in different parallel architectures and was used to execute the parallel Ada code.

SECTION II

GUIDANCE, NAVIGATION, AND CONTROL (GN&C) BENCHMARKING

1. PHILOSOPHY OF BENCHMARKING

When trying to compare the performance of one microprocessor to another it's hard to select a "best" processor, due to the many variables, and to keep from comparing "apples and oranges."

Typically advertising literature quotes the speed of a processor in MIPS for a classical benchmark. The problem with these benchmarks is that they usually represent only one type of computation, in a small amount of memory, thus do not truly represent the performance of the processor for all applications. For example, the small memory size often can fit in the fast cache memory of some processors, which leads to excellent results, but does not represent problems that occur for most real applications that exceed the cache size.

Therefore, the only true way to know the performance for your application for various processors is to run your entire application on the processors, versus relying on quoted benchmarks, which may have no relevance to your application. However, this can be costly and time-consuming to do. While attempting to do this for our application (GN&C), we found there are still many "apples and oranges."

a. All Levels of System Impact Performance

Figure 1 shows a hierarchical list of possible variables that can have a large impact on a measured performance result. The microprocessor is just one piece of an overall system. Thus the overall efficiency and speed of the system is what is important, not just the speed of the microprocessor. It is also important that the system used in the performance evaluation closely model the actual application of the embedded system.

(1) Microprocessor

The microprocessor design obviously impacts the design of all the other layers (see Figure 1). A microprocessor can be driven at different clock speeds, which directly effects processing speed. In general, assuming other system features such as memory can keep up, as the clock speed of a processor increases, the speed of the benchmark will increase proportionately. Even in the short time of this project, the microprocessor vendors have come out with new versions of their microprocessor at faster and faster clock speeds. If the processor is upwardly compatible in software, its speed can be proportionately increased. Sometimes newer versions of processors offer other enhancements, such as the parallel execution of some instructions, which nonlinearly increase performance and cannot be extrapolated. Therefore,

the performance must be measured by running the benchmark on the new processor itself. If a newer processor were not upwardly compatible in software, the upper layers in Figure 1 would have to be redesigned, thus rebenchmarked.

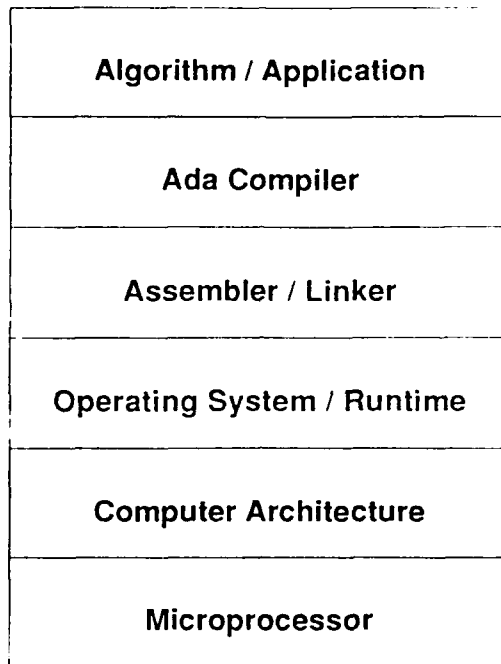


Figure 1. All Levels of System Impact Performance

The efficiency of many new processors is gained by deep pipelines (see Figure 2). As long as the pipeline remains full of useful instructions, good performance is maintained. However, events such as branches or external memory access can cause a pipeline stall, whereby the pipeline must be flushed and refilled with useful instructions. These delays greatly impact performance. The efficiency that can be gained by the pipeline of a processor is dependent on the other layers of the system (Figure 1) such as cache and memory management, the compiler, assembler, and run-time systems use of the pipeline architecture. The global and local branches required by the algorithm also impact pipeline efficiency. To limit pipeline stalls, some architectures offer parallel execution of the two sides of a branch.

Figure 2 shows the benefit of having an entire instruction set execute in the same number of instruction cycles, which is a goal of RISC processors. This leads to a more efficient pipeline (no wasted cycles) and simpler pipeline control circuitry.

Another concern of microprocessors is scalability. If a processor is scalable, its timing and control is independent of the clock speed. Therefore, for non-scalable processors, if the clock speed is increased above a certain point, the timing and control circuitry has to be

redesigned. This often leads to software incompatibility and a nonlinear performance increase due to clock speed. RISC processors tend to be more scalable than CISC.

(2) Computer Architecture

Beside the microprocessor (see Figure 1), the computer architecture in which it is designed can have a big impact on performance. For example, the size and speed of the random access memory (RAM) will have an impact. No matter how fast a processor is, if it cannot be fed fast enough by memory and has to wait a long time for instructions or data, its performance will be slowed. Many designs use fast cache memory to interface between RAM and the processor to keep the processor's pipeline fed. The size of the cache affects the efficiency of cache hits. A large cache increases the probability that a hit will occur; however, if a hit does not occur, the cache must be flushed and refilled with new memory, which takes longer for a bigger cache. The optimum cache size varies depending on application.

To smartly manage these memory transfers, some architectures have memory management units (MMU). Again, the efficiency of the MMU is determined by the application.

The two basic computer architectures styles are Von Neumann and Harvard. Von Neumann architecture has one external bus that is used for both data and instructions, while a Harvard architecture has two separate buses, one for importing instructions and one for data. In an ideal situation, the Harvard architecture can double the memory throughput. However, it adds the complexity and cost of an extra 32-bit bus on the chip. The data/instruction mix determines the relative efficiency of the two different architectures and therefore, is application dependent.

Most embedded applications have to fit the entire application in RAM or read only memory (ROM), while a traditional computer has other resources such as a hard disk. Other resources such as math coprocessors (and its clock speed) or hardware implemented functions can have an impact. Today with better manufacturing and higher gate density on a chip, more of the computer architecture is being moved on-chip (that is, on-chip coprocessors, cache, MMU, and parallel execution). With each new release of a microprocessor family, more features are added on-chip, increasing the performance, often nonlinearly and application dependent, making extrapolation of previous benchmarks more difficult.

Finally, for benchmarking, the computer architecture needs to reflect the way it would be used in the embedded application, and hopefully in an optimized manner.

(3) Operating System/Run-Time

Above the hardware layer is the software environment (Figure 1). Traditional computers run an operating system (for example, UNIX, VMS, or MS-DOS), while embedded systems have a run-time environment. The

efficiency of the run-time environment or operating system will impact the overall system efficiency. Again, this run-time or operating system should match what would be used in the final embedded application.

(4) Assembler/Linker

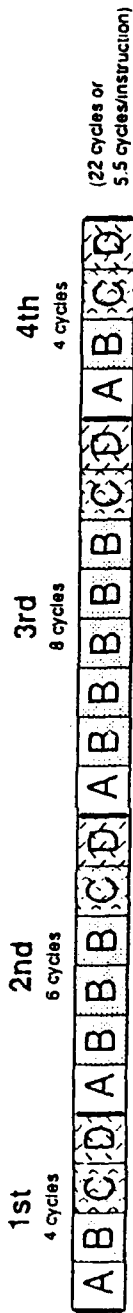
The assembler/linker is software that converts the assembly language code to machine language and links together the code into the final image to be downloaded to the machine. The efficiency of this software (that is, smart use of the processor's ISA) will greatly impact performance. For example, some RISC processors use no operation (NOOPs) instructions for timing. Instructions that last more than one instruction cycle (for example, a branch) will have a NOOP to ensure the next real instruction does not effect the resource the previous instruction was using. However, certain instructions, which do not use conflicting resources, could be implemented during the NOOP without causing timing problems. A "dumb" assembler will add NOOPs everywhere needed, which will result in wasted instruction cycles. A "smart" assembler will look for ways to reorganize the instructions to replace the NOOPs with useful instructions, thus removing wasted instruction cycles and increasing performance. The use of NOOPs for timing tends to make a RISC architecture nonscalable in software, but not hardware. Therefore, when a new version of a processor is released with a faster clock speed, the software tools (assembler/linker) will have to be modified to ensure the NOOPs are inserted properly and efficiently to control the timing and use of resources.

Another important aspect of the assembler/linker is how well it uses lower system features efficiently such as the math coprocessor and cache.

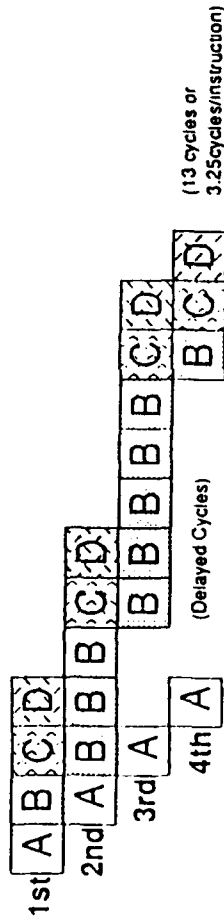
(5) Ada Compiler

The maturity of the Ada compiler has a high impact on performance. Generally the early versions of an Ada compiler will implement the Ada language properly, but not necessarily efficiently. With time, newer versions of the compiler will be offered which will make better use of the assembler, operating system, computer architecture, and microprocessor. Many Ada compilers can perform different levels of optimizations. The ACEC Reader's Guide gives details of this partial list of optimizations: common subexpression elimination, folding, loop invariant motion, strength reduction, dead code elimination, register allocation, loop interchange, loop fusion, test merging, Boolean expression optimization, algebraic simplification, order of expression evaluation, jump tracing, unreachable code elimination, use of machine idioms, and packed Boolean array logical operators. These optimizations can improve the execution speed and/or decrease the code size, but usually increase the compilation time. The philosophy followed in this project was to select all optimization options that worked correctly. All combinations of optimizations were attempted, and the combination that resulted in the best execution time was reported.

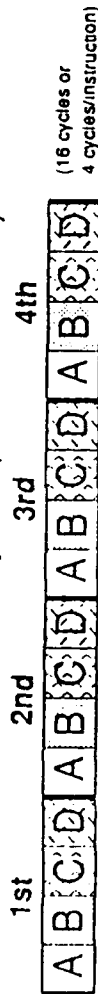
Complex Instructions (variable number of cycles per instruction)



Pipeline Execution



Reduced Instructions (fixed number of cycles per instruction)



Pipeline Execution



Figure 2. Pipeline Example

To obtain the fastest possible version of the code, the PRAGMA SUPPRESS feature of Ada was used to suppress run-time checks (that is, index_check, discriminate_check, length_check, range_check, division_check, overflow_check, elaboration_check, and storage_check). During software design, these run-time checks should not be turned off. However, after the code has been thoroughly debugged and is ready for embedded use, the checks are removed because they add no value and slow down the processing.

Since Ada was the language used in all applications within this project, it was the only one benchmarked. Another source of variability can be introduced through different languages, such as C, FORTRAN, PASCAL, LISP, and OCCAM.

(6) Algorithm/Application

The top level of Figure 1 is the algorithm that is dependent on the application. As with all these levels of design, the algorithm can change greatly, throughout a project, as requirements change. Typically, the final algorithm and requirement are not determined until the end of a project. However, the algorithm used in benchmarking must represent the final application as much as possible and is another source of variability.

How closely the algorithm models the hardware architecture can also impact performance. For example, in Ada, the precision of the REAL or floating point variables can be specified. If this does not match the precision of the hardware, the manipulation of REAL numbers will be inefficient.

All the candidate systems were modeled as closely as possible to the final GN&C application, knowing there are still many variables that could improve or hinder each system.

b. Real Requirement

In determining a "best" processor to meet your applications needs, the fastest processor may not be the only acceptable processor. For our GN&C application, the execution time requirement is less than 10,000 microseconds (or 100 Hz cycle time). Several processors may meet this requirement. Obviously the more margin the better, in case of future upgrades or changes in the requirements. Given that several processors meet the applications speed requirement, other factors such as cost and software support will come into play. Therefore, when developing a benchmark to represent the application, the maximum allowable execution time requirement for the benchmark should be determined.

c. Measurements

Benchmark results are presented in various forms: raw time, MIPS, and floating point operations per second (FLOPS). Actual MIPS are hard to calculate. The implication is that you would have to count the actual number of instructions executed for the benchmark for each processor. Also

"instructions per second" is not a fair comparison between processors, especially for CISC and RISC processors. It may take several RISC instructions to do the equivalent work of a single CISC instruction. A more meaningful measure would be equivalent work per second or benchmark per second. To generalize this measure over several processors and benchmarks, "VAX-MIPS" are used. The Digital Equipment Corporation (DEC) MicroVAX II is known to process on the average around 1 MIPS (actually ~0.9 MIPS). Therefore, the benchmark is executed on both the VAX and the processors in question. The speed of each processor is then calculated relative to the speed of the VAX. For example, if the VAX time for a benchmark was 100 microseconds and a processor time was 20 microseconds, the processor would be 5 times faster, therefore, 5 MIPS for the benchmark.

The VAX-MIPS speed calculated for a benchmark is valid only for that specific benchmark and specific combination of system features. The VAX-MIPS speed should not be generalized as a speed for the processor for all applications and system combinations.

Most microprocessor families have versions that run at various clock frequencies, and the processors are often not benchmarked on a system with their maximum clock frequency. The clock speed has a direct impact on VAX-MIPS, although not always linear as discussed in Section II.1a(1). To reduce "apples and oranges" comparisons, two other calculations can be made:

- Relative speed (VAX-MIPS) normalized to a common clock speed. We used 20 MHz, a speed compatible for most systems. For example, if a benchmark was measured to be 5 VAX-MIPS on a 25 MHz processor, the normalization to 20 MHz would be $5 \cdot (20/25) = 4$ VAX-MIPS at 20 MHz.
- Relative speed (VAX-MIPS) extrapolated to the maximum known clock speed for the processor.

These calculations help level the playing field when comparing processors. However, clock speed is not always directly proportional to processor performance; therefore, these normalizations are just an estimate. The only true benchmark could be obtained through benchmarking on systems with the exact clock speeds.

The averaging of the results of several different benchmarks is not statistically valid. However, we have also calculated the average results for a rough estimate of processor performance.

For accuracy in measuring the execution time, we used the ACEC benchmarking tools, discussed further in Section II.3.

2. BENCHMARKS FOR GN&C APPLICATIONS

The first step in an evaluation is to define the standard or requirements. In the case of benchmarking processors, the standard is

defined by the algorithm of the benchmark. The benchmark should model the application as close as possible. For GN&C applications we have the CAMP, ACEC, PIWG, and OGLI Ada benchmark suites. Because the CAMP, ACEC, and PIWG benchmarks have been run on some processors, but not all, these results are not presented. We developed our own in-house benchmark, OGLI, which represents the currently used algorithms. The results for the OGLI benchmarks are presented in Section II.5.

a. CAMP

The CAMP benchmarks are guidance and navigation benchmarks in Ada developed during the CAMP program. This program also included the development of over 500 reusable Ada software parts with a library cataloging system, and an expert system to aid in using the parts. (See Reference 2.)

The CAMP benchmarks were used for evaluation during the GISA program, as discussed in Section I.2c. The processors compared included a MIL-STD-1750A, a Motorola 68020 (CISC), a MIPS R3000 (RISC), and an Intel 80960 (RISC). This evaluation demonstrated that the two 32-bit RISC architectures far out-performed the 16-bit MIL-STD-1750 and the 32-bit 68020 CISC. The R3000 was the better performer of the two RISC processors. (See Reference 1.)

The CAMP benchmarks are a good representation of modern, Ada GN&C benchmarks. However, we used the OGLI benchmarks which more closely represent our current applications.

b. ACEC

The ACEC benchmarks contain over 1,000 Ada benchmarks representing all aspects of the Ada language. These benchmarks are good for testing fine details of a compiler; however, they do not necessarily represent our specific GN&C application. Most of the benchmarks are tiny and only measure one feature of the language. A few larger applications, such as a Kalman filter, are included in the benchmarks. The task of running numerous benchmarks is very time-consuming, but it is automated by batch files. For the ACEC benchmarks, it is important to study the benchmarks to understand what is really being measured.

While this project does not report any ACEC benchmark results, the ACEC benchmarking tools used are discussed in Section II.3. (For more information about the ACEC benchmark, see References 3 and 4.)

c. PIWG

The PIWG of the Association of Computer Manufacturers (ACM) special interest group on Ada (SIGAda) has a series of general purpose benchmarks designed to measure the performance of a processor and compiler on many aspects of Ada. Copies of the benchmarks are free, provided the copies are not used for commercial advantage and copyright credit is given to ACM SIGAda. The PIWG benchmarks were not used in this evaluation; however, some

of these benchmarks were used in the GISA evaluation. The particular benchmarks used included the Drystone, A00091, an integer benchmark that does string operations; the Whetstone, A00093, which contains 15 percent floating point operations; the procedure calling benchmarks, P00001-P00013; and eight tasking benchmarks. (The results of these benchmarks can be found in Reference 1.)

d. OGLI

The OGLI benchmark contains a guidance law designed for an air-to-air missile. This guidance law is representative of the code used for our in-house research at the start of this project. As shown in Figure 3, guidance receives inputs from the seeker regarding the change in state of the target (that is, elevation and azimuth line of sight rate, range rate, and range). The Inertial Navigation System (INS) provides the current state of the missile (inertial accelerations) to guidance. Guidance then calculates the commanded inertial accelerations necessary to optimally fly to the target and passes the commands to the autopilot. (See Reference 5.)

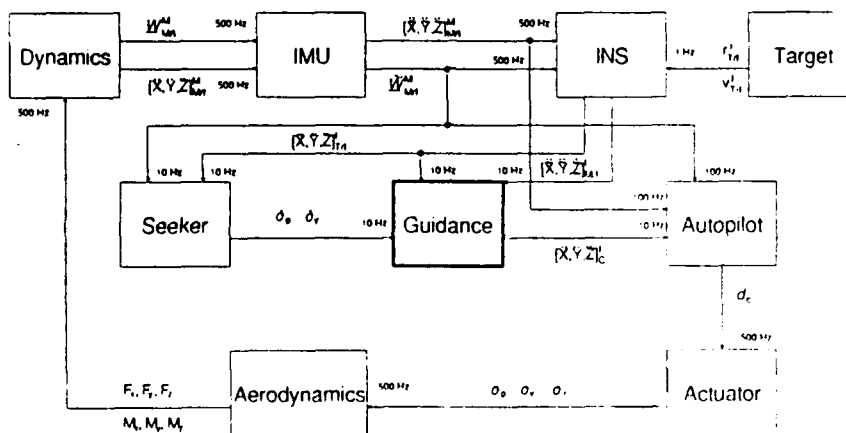


Figure 3. Optimum Guidance Law Implementation

The OGLI benchmarks are divided into four main benchmarks representing the four phases of flight. The first phase is Full_Midcourse, the longest phase in which the guidance law is optimized to maintain maximum energy. When closer to the target the guidance law switches to Ramping_Midcourse, in which it smoothly adjusts its heading for an anticipated straight line collision course with the target. The next phase is called Transition, in which the autopilot tries to smoothly transition from the midcourse commands to the commands of the next phase, known as the Terminal. The Terminal phase is the end game, and the guidance law switches to augmented proportional navigation.

The speed requirements are less than 100,000 microseconds (10 Hz) for the midcourse phases and less than 20,000 microseconds (50 Hz) for the terminal phase. As a safe margin for error, our goal is for all phases to

less than 10,000 microseconds (100 Hz). In the future, greater performance will be needed for some modern techniques. For example, we would like to use several different guidance laws and have the capability to smartly select the optimum commands in real time.

3. BENCHMARKING TOOLS

The ACEC benchmarking tools provide the timing loop code to "wrap around" the benchmark algorithms and thus accurately measure the execution time. The benchmarks are nested in two loops:

```
OUTER LOOP
  start_time_hack;
  INNER LOOP
    benchmark;
  END INNER;
  stop_time_hack;
END OUTER;
```

The ACEC tools first measure the overhead execution time for the inner loop and check the accuracy of the clock. Via test runs of the benchmark, the ACEC tools determine the number of inner and outer loops required to have statistically valid results, based on the means and variances of the execution time for each inner loop of the benchmark. The tools output the mean execution time (minus the timing loop overhead), the variance, and the number of inner and outer loops.

These tools are written in Ada and allow a processor independent version of the OGLI benchmarks, which were easily transported to each of the processors. (See References 3 and 4.)

4. CANDIDATE MICROPROCESSORS

In this section, each microprocessor and Ada compiler that was considered as a candidate for evaluation is discussed. This list includes the RISC and CISC processors we were considering for future embedded use. This list does not include all RISC and CISC processors nor all Ada compilers for each processor. At the time of this evaluation, most RISC processors only had one Ada compiler vendor, while the older CISC, with a larger commercial support, have multiple sources for Ada compilers. Due to limited funds and time, not all systems could be evaluated. This evaluation is a continual effort, with the results continually updated. This paper is just a "snapshot" in time.

a. DEC MicroVAX II (CISC)

The DEC MicroVAX II system was used as the base line for comparison (as discussed previously in Section II.1c.) The MicroVax II processor, KA630-AA, is not being considered for embedded use and is a classic CISC architecture. The MicroVAX II is recognized to have a speed in general of 1 MIPS (actually 0.9). The speed of the benchmarks on the candidate processors

relative to the MicroVAX is used to calculate the VAX-MIPS for each processor.

The VAX was configured with an 8-megabyte (Mbyte) RAM, a math coprocessor, and the VMS 5.2 operating system.

The DEC Ada compiler Version 2.0 was used. This compiler offers optimization with respect to time and size. All optimization combinations were benchmarked. The time optimization resulted in not only the best execution time but also decreased the size compared to no optimizations.

b. Intel 80386 (CISC)

The Intel 80386 is a popular 32-bit CISC found in commercial personal computers, and therefore, has a large software base. It has an upwardly compatible family along with the 8086, 80286, and the new 80486. The 80386 has a maximum clock speed of 40 MHz. The 80486 soon will be offered at 50 MHz. The 80386 DX has a full 32-bit central processing unit (CPU) and can address up to 4 gigabytes (Gbytes) of physical memory and 64 terabytes (Tbytes) of virtual memory. The 80386 interface can support the external math coprocessor with 32-, 64-, and 80-bit formats.

Two different 80386 configurations were benchmarked. The first had a clock speed of 25 MHz, hosted on a Diversified Technology's CAT980A board, with an 8-Mbyte RAM (two wait state), 32-kilobyte (kbyte) cache, Intel 80387 coprocessor, and IBM DOS Version 3.30.

The second configuration had a clock speed of 40 MHz, hosted on Bell Computer System's PAT38PX board, with an 8-Mbyte RAM, 256-kbyte cache (zero wait state), Intel 80387 coprocessor, and MS-DOS Version 4.01.

A 80286 system was also benchmarked with an 8-MHz clock, hosted on a Zenith 2-248 computer, with a 3.2-Mbyte RAM, Intel 80287 coprocessor, and MS-DOS Version 3.3. However, compilation of the OGLI benchmarks on this configuration was not completed due to memory limitations.

The Ada compiler for both 80386 configurations was Alsys Version 4.5. Both used the same executable code compiled on the 25 MHz machine. The Alsys compiler offers four command line optimizations: tasking, calls, reduction, and expression. Since OGLI does not use Ada tasking, tasking was turned off. The other three optimizations were first used in the normal mode (turned off), and OGLI compiled successfully. Next, the "calls" optimization was set to "inlined," which replaces (inlines) all calls to subroutines with the actual subroutine code. With this optimization selected, the compiler never "returned" from compiling. The memory was probably overloaded due to a combinational explosion of calls. Next, the reduction and expression optimizations were set to "extensive." This combination of options compiled successfully. However, during execution, it did not run successfully and generated an exception or error. Therefore, the performance results reported in this paper are for compiled code with tasking turned off and no other optimizations.

The Ada compiler for the 80286 was Meridian Version 2.0. This compiler was unable to complete compilations of OGLI due to a memory limitation of the computer.

c. Motorola 68040 (CISC)

The Motorola 68000 family are CISC with a very large commercial base. The family consists of the 68000, 68008, 68010, 68020, 68030, and 68040, which are upwardly compatible.

The 68020 is a full 32-bit, Harvard architecture, microprocessor with 101 instructions. Its features include cache control registers to support external data and instruction caches, a 32-bit direct address range, a 4 Gbyte virtual address range, 8 data and 8 address 32-bit registers, and a 20-MHz clock. Coprocessors, MMU, and caches are supported via external chips.

The on-chip enhancements of 68030, compared to the 68020, includes a MMU and 256-kbyte data and instruction caches. The MMU and caches operate in parallel with the CPU.

The on-chip enhancements of 68040, compared to the 68030, include a floating point coprocessor (three stage pipeline); a new six stage, dual execution, single-cycle pipeline for the CPU; two MMUs, each with their own translation lookaside cache; 4-kbyte data and instruction caches; a bus control unit; and a 25-MHz clock. All the processing units operate in parallel.

Currently, the OGLI benchmarks have not been run on the 68000 family. The 68020 was benchmarked in the GISA program versus a MIPS R3000, a MIL-STD-1750, and an Intel 80960 using the CAMP benchmarks. (For more information see Reference 1.)

d. MIPS R3000 (RISC)

The MIPS Computer Systems R3000 is a 32-bit RISC processor. Its instruction set was originally developed at Stanford University and meets the requirements of DARPA's CSALI. The family includes the original R2000, the R3000 (CMOS) with a maximum clock of 33 MHz and the R6000 (ECL) with a maximum clock speed of 60 MHz. The R4000 CMOS is anticipated soon. The family is code compatible. The architecture is scalable in hardware, but non-scalable in terms of software, as mentioned in Section II.1a(4). The software is non-scalable since NOOPs are needed for timing. Therefore, the assemblers and reorganizers may have to be modified as the clock frequency is increased above certain thresholds.

The R3000 is a full 32-bit Von Neumann architecture. Its features include a five-stage CPU pipeline, thirty-two 32-bit registers, and MMU with a 64-entry translation lookaside buffers to support 4 Gbytes of virtual memory. It can also have cache control registers to support

external instruction and data caches up to 64 kbytes each. The R3000 supports up to three external coprocessors.

The benchmark configuration is a R3000 system built by Westinghouse during the GISA program. The R3000 board resides in a VME rack, hosted by a MicroVAX II. The clock speed is set at 14 MHz. The board has 1-Mbyte RAM and dual caches (64 kbytes for instruction and 64 kbytes for data). The dual external caches add the benefits of a Harvard architecture. With separate caches, the code is more likely to be reused (and not flushed as often) than if the instructions and data were mixed on one cache. The math coprocessor resides on a separate chip, the R3010. The run-time environment was developed specifically for the GISA program, designed for real-time, distributed processing. (For more information see Reference 1.)

The Ada compiler is the InterAct 1.0, hosted on the MicroVax II. This compiler optimizes the code by default.

e. Sun SPARC (RISC)

The Sun SPARC is a 32-bit RISC processor developed for use in Sun workstations. The maximum clock speed for the CMOS version is 40 MHz, while the ECL version can go up to 80 MHz.

The SPARC is a Von Neumann architecture. It has a large register bank of 128 registers. By using the concept of overlapping register windows, SPARC has a performance advantage. The windows can act like a local variable cache, reducing the amount of memory traffic (number of parameters that must be saved or restored) during subroutine calls.

The configuration used for OGLI benchmarking included a Sun-4 SPARCstation 1, with an 8-Mbyte RAM, 128-kbyte cache, math coprocessor, and the UNIX-4 operating system.

The Ada compiler was Verdix VADS Version 6.0(G). All optimizations were used. The Verdix User Guide (Reference 6) lists the following applicable optimizations: code straightening, constant folding, copy propagation, strength reduction, redundant branch and range check elimination, common subexpression elimination, hoisting loop invariant computations and range checks, limitation of assignment to unused local variables, address simplification, local scalar and access variables allocated in registers, loop variables allocated in registers, parameters passed in registers, code generation for math coprocessors, and target specific peephole optimization. (However, with all these optimizations, one of the benchmarks calculated an incorrect answer, but the cause has not been found.)

f. Motorola 88000 (RISC)

The Motorola 88000 is another 32-bit RISC processor. Currently the maximum clock speed available is 33 MHz. The 88000 is implemented in a three chip set--the 88100 is the CPU chip and there are two 88200

cache/memory management units. The 88000 has a Harvard architecture and 32 registers. The 88100 has a floating point unit on-chip, with a five-stage add pipeline and a six-stage multiply pipeline.

The configuration used for OGLI benchmarking was part of a Data General AVIION 300 workstation based on the 88000, at a clock speed of 20 MHz, with a 12-Mbyte RAM, dual caches (16-kbyte instruction and 16-kbyte data), and the DGUX 4.3 UNIX operating system.

The Ada compiler used was the Telesoft Telegen2 Version 4.0. For benchmarking, we compiled with the default settings, which includes all safe optimizations. When compiling with the global optimizations, the system ran out of paging space due to the large size of the program. This probably could have been avoided if the OGLI benchmarks were split into several small packages, instead of one big procedure.

Telesoft offers two levels of optimizations. The User's Guide (Reference 7) shows Level 1 including subprogram inlining, common subexpression elimination, dead code elimination, local value propagation, range calculation and propagation, constant folding and propagation, check elimination, and subprogram interface optimization. Level 2 includes lifetime minimization, tail recursion elimination, loop-invariant code motion, dead-store elimination, induction variable identification, and copy propagation.

g. IBM System 6000 (RISC)

The IBM System 6000 is a workstation built around RISC concepts. It uses a Performance Optimization with Enhanced RISC (POWER) architecture. The family of POWER processors have different names for each clock speed. The IBM System 6000 used for this evaluation is workstation model 530, which has the 25-MHz SGR2564 processor. Other models have clock speeds of 20 and 30 MHz. The fastest advertised processor with the POWER architecture has a clock speed of 41.6 MHz.

The processor is designed to do multiple operations in parallel. Up to four instructions can be executed in one clock cycle, provided they do not need to use the same resources. To keep the parallel operations fed, there is an 8-kbyte instruction cache and 64-kbyte data cache. The data cache is split into four banks for each type of operation. The cache memory and the floating point processor are all on-chip. The parallelism built into this design allows for such features as zero-cycle branch operations.

The system used for evaluation contains 64-Mbytes of RAM. The UNIX operating system is used. The system is built around the IBM microchannel input/output bus.

The Ada compiler for the IBM System 6000 family is the IBM Ada/6000, Version 2.0. The user's manual (Reference 8) lists three levels of optimization: normal, optimized, and highly optimized. The optimized level does such optimizations as elimination of unreachable code, common

subexpressions, parameter substitution, constant folding and propagation, and strength reduction. The highly optimized level does the more risky optimizations like the inlining of calls. For the OGLI benchmarks, we found the "optimized" level produced the fastest and smallest code. The inlining of code for OGLI turns out to be less efficient.

h. Intel i960 (RISC)

The Intel i960 RISC processors have a Von Neumann 32-bit architecture with 32 registers. The i960 is marketed as an embedded processor. All of its simple instruction set fits within a 32-bit boundary which aids pipelining. On the military version, the 80960MC has a 512-byte instruction cache and floating point capabilities on-chip.

The i960 processors have not been benchmarked with the OGLI code. The main competitor of this processor appears to be the R3000. During the GISA program (discussed in Section I.2c) the i960 was benchmarked against a R3000, 68020, and MIL-STD-1750. (See Reference 1.)

i. Texas Instruments (TI) TMS320C30 (Signal)

The TI TMS320C30 (or C30) is a floating-point, digital signal processor. It is probably the first signal processor to have an Ada compiler, which implies it is capable of general purpose applications too. This family includes the C30 and C40, which both have a maximum clock speed of 40 MHz.

The C30 on-chip features include 40- or 32-bit CPU (floating point and integer), 64 by 32-bit instruction cache, two 1K by 32-bit single-cycle dual-access RAMs, 4K by 32-bit single-cycle dual-access ROM, 8 extended precision registers, 8 auxiliary registers, 12 control registers, two external ports, and two 32-bit timers.

The C40 on-chip features include all the C30 features plus direct memory access coprocessor, 512-byte instruction cache, 12 extended precision registers, 14 control registers, six communication ports, and two 4-kbyte RAMs and 16-kbyte ROM that allow two data accesses in a single cycle.

The C30 configuration used for OGLI benchmarking had a clock speed of 16 MHz. Most of the benchmark could fit in zero wait state memory; however, some spilled over into two wait state memory, which is a handicap.

Tartan makes the Ada compiler for the C30; however, the version or optimizations used for benchmarking were not recorded.

j. INMOS Transputer T800 (Parallel)

The INMOS transputer T800 is a processor designed for parallel processing. The T800 has four bidirectional, high speed, serial input/output (I/O) ports designed to easily connect to T800s or other

transputers. A goal with the transputers is low cost, making super parallel computers affordable and easily to build. The transputer is marketed as a RISC processor. The T800 is a full 32-bit processor, with a 64-bit floating point unit and 4-kbytes of RAM on-chip. The T800 has a maximum clock speed of 30 MHz. Unlike most RISC processors, the T800 has a small set of 6 registers. The transputer family includes the upwardly compatible T400, T800, and T805.

A single transputer T800 was used for OGLI benchmarking. In Section III, multiple T800s are used in a parallel implementation of the OGLI code. For benchmarking, the T800s reside on a "motherboard" that hold up to seven T800s. The motherboard is hosted in a Zenith Z-248 personal computer. The main processor has 4 Mbytes of RAM and a clock speed of 25 MHz, while each of the other six processors has 1 Mbyte of RAM and a clock speed of 20 MHz. The T800 has specialized run-time software called ISERVER.

The transputer has its own high level language, called OCCAM, which was developed together with the transputer, specifically for parallel processing. Therefore, OCCAM runs very efficiently on the transputer, just as an assembly language would.

Alsys developed the first (and only) Ada compiler for the transputer. For benchmarking, Version 4.4 was used. This is an early version of the compiler (in the first year), therefore, still has plenty of room to mature. The compiler offers the same optimizations as the Alsys compiler for the 80386, discussed in Section II.4b. The benchmark was compiled with the default level of optimizations. Alsys used the same user interface for both the 80386 and transputer, which was "user friendly" and made the benchmarking batch file portable.

A small portion of the OGLI benchmarks (called Step 1) was converted from Ada to OCCAM for comparison. The results are presented in the next section.

5. RESULTS AND ANALYSIS

The results for the OGLI benchmarks are presented in Tables 1, 2, and 3 and Figures 4 through 8. These results represent performance for a specific GN&C application and cannot be generalized as a universal performance. The results are presented for three measures: execution speed, compile time, and code expansion.

TABLE 1. OGLI BENCHMARK RESULTS

Benchmark Processor	Ada Compiler	Clock Speed (MHz)	Exec. Time (usec)	Speed to uVAX II	Speed Relative to 20 MHz	Speed Normalized to Max Clock	Max Clock (MHz)
Full_Midcourse							
uVAX II	DEC 2.0	--	27605.7	1.00	----	----	--
SPARC	Verdix 6.0	20	3252.1	8.49	8.49	34.0	80 (ECL)
88000	TeleSoft4.0	20	3519.8	7.84	7.84	12.9	33
R3000	InterAct1.0	14	5270.0	5.24	7.48	22.4	60 (R6000)
IBM6000	IBM 2.0	25	1264.9	21.82	17.46	36.3	41.6
C30 2wait*	Tartan	16	3884.9	7.11	8.89	22.2	50
T800	Alsys 4.4	25	8986.3	3.07	2.46	3.7	30
80386	Alsys 4.5	25	262448.9	0.11	0.09	0.2	50 (486)
80386	Alsys 4.5	40	20530.4	1.34	0.67	1.7	50 (486)
Ramping_Midcourse							
uVAX II	DEC 2.0	--	27321.2	1.00	----	----	--
SPARC*	Verdix 6.0	20	2799.3	9.76	9.76	39.0	80 (ECL)
88000	TeleSoft4.0	20	3465.2	7.88	7.88	13.0	33
R3000	InterAct1.0	14	5166.7	5.29	7.55	22.7	60 (R6000)
IBM6000	IBM 2.0	25	1243.2	21.98	17.58	36.6	41.6
C30 2wait	Tartan	16	3969.5	6.88	8.60	21.5	50
T800	Alsys 4.4	25	8950.2	3.05	2.44	3.7	30
80386	Alsys 4.5	25	263922.8	0.10	0.08	0.2	50 (486)
80386	Alsys 4.5	40	20242.4	1.35	0.67	1.7	50 (486)
Transition							
uVAX II	DEC 2.0	--	25965.6	1.00	----	----	--
SPARC	Verdix 6.0	20	2895.4	8.97	8.97	35.9	80 (ECL)
88000	TeleSoft4.0	20	3247.3	8.00	8.00	13.2	33
R3000	InterAct1.0	14	4770.0	5.44	7.78	23.3	60 (R6000)
IBM6000	IBM 2.0	25	1171.0	22.17	17.74	36.9	41.6
C30 2wait	Tartan	16	3620.2	7.17	8.96	22.4	50
T800	Alsys 4.4	25	8087.8	3.21	2.57	3.9	30
80386	Alsys 4.5	25	267486.9	0.10	0.08	0.2	50 (486)
80386	Alsys 4.5	40	18663.9	1.39	0.70	1.8	50 (486)
Terminal							
uVAX II	DEC 2.0	--	4785.0	1.00	----	----	--
SPARC	Verdix 6.0	20	789.8	6.06	6.06	24.2	80 (ECL)
88000	TeleSoft4.0	20	861.9	5.55	5.55	9.2	33
R3000	InterAct1.0	14	863.3	5.54	7.92	23.8	60 (R6000)
IBM6000	IBM 2.0	25	259.1	18.47	14.77	30.7	41.6
C30 2wait	Tartan	16	952.3	5.02	6.28	15.7	50
T800	Alsys 4.4	25	2173.0	2.20	1.76	2.6	30
80386	Alsys 4.5	25	300039.4	0.02	0.01	0.04	50 (486)
80386	Alsys 4.5	40	5639.6	0.85	0.42	1.1	50 (486)

TABLE 1. OGLI BENCHMARK RESULTS (Concluded)

Overall Processor	Clock Speed (MHz)	Average Relative Speed	Average Range	Average for 20 MHz	Range for 20 MHz	Average for Max Clock	Range for Max Clock	Max Clock (MHz)
uVAX II	--	1.00	1.00-1.00	----	-----	----	-----	--
SPARC	20	8.32	6.06-9.76	8.32	6.06-9.76	33.3	24.2-39.0	80 (ECL)
88000	20	7.32	5.55-8.00	7.32	5.55-8.00	12.1	9.2-13.2	33
R3000	14	5.38	5.24-5.54	7.68	7.48-7.92	23.1	22.4-23.8	60 (ECL)
IBM6000	25	21.11	18.5-22.2	16.89	14.8-17.7	35.1	30.7-36.9	41.6
C30	16	6.55	5.02-7.17	8.19	6.28-8.96	20.5	15.7-22.4	50
T800	25	2.88	2.20-3.21	2.31	1.76-2.57	3.5	2.6- 3.9	30
T800-3 par.		5.06	3.96-5.51	----	-----	6.1	4.8- 6.6	30
80386	25	0.08	0.02-0.11	0.07	0.01-0.09	0.2	0.04-0.2	50
80386	40	1.23	0.85-1.39	0.62	0.42-0.70	1.6	1.1- 1.8	50

TABLE 2. STEP 1 BENCHMARK RESULTS

Processors	Time(usec)	VAX MIPS	Normalized to 20 MHz
uVAX II	2683.9	1.00	-----
SPARC (20MHz)	447.4	6.00	6.00
88000 (20MHz)	384.6	6.98	6.98
R3000 (14MHz)	374.4	7.17	10.24
80286	25603.6	0.10	-----
T800 (Occam-25M)	202.0	13.29	10.63
T800 (ada-25MHz)	2393.6	1.12	0.90

TABLE 3. CODE EXPANSION AND COMPILE TIMES

Processor	Host	Compiler	Size (bytes)	Code Expansion	Compile Time	Relative to uVAX
uVAX II	----	DEC 2.0	59392	1.00	0:08:04	1.00
SPARC	----	Verdix 6.0	303104	5.10	0:05:43	0.71
88000	----	TeleSoft 4.0	478588	8.06	0:04:50	0.60
R3000	uVAXII	InterAct 1.0	228352	3.84	0:40:51	5.07
IBM6000	----	IBM 2.0	122094	2.06	0:02:43	0.34
T800	Z248(PC)	Alsys 4.4	110711	1.86	1:22:41	10.37
80386	----	Alsys 4.5	172018	2.90	0:08:50	1.10

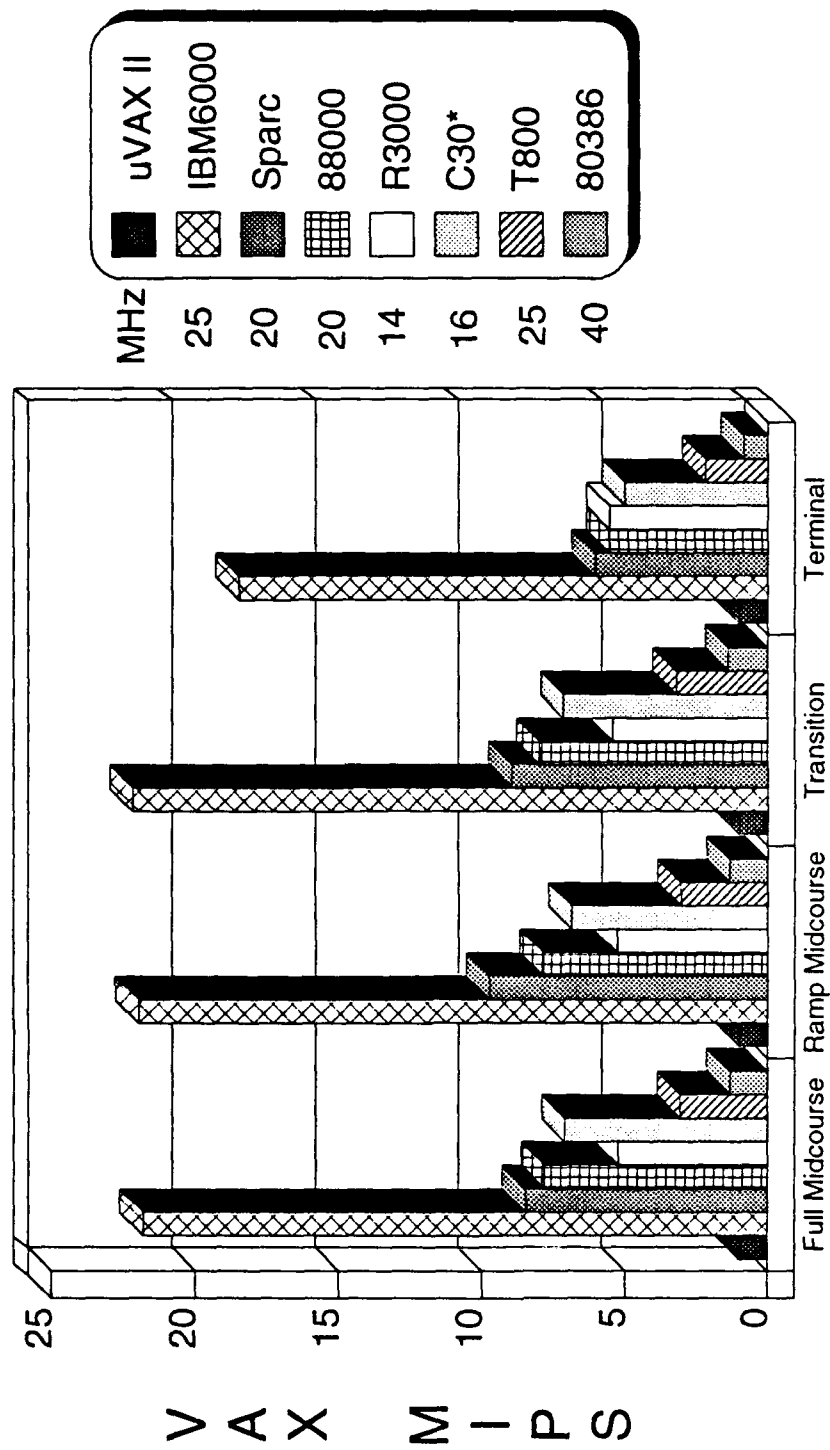


Figure 4. OGLI Benchmarks

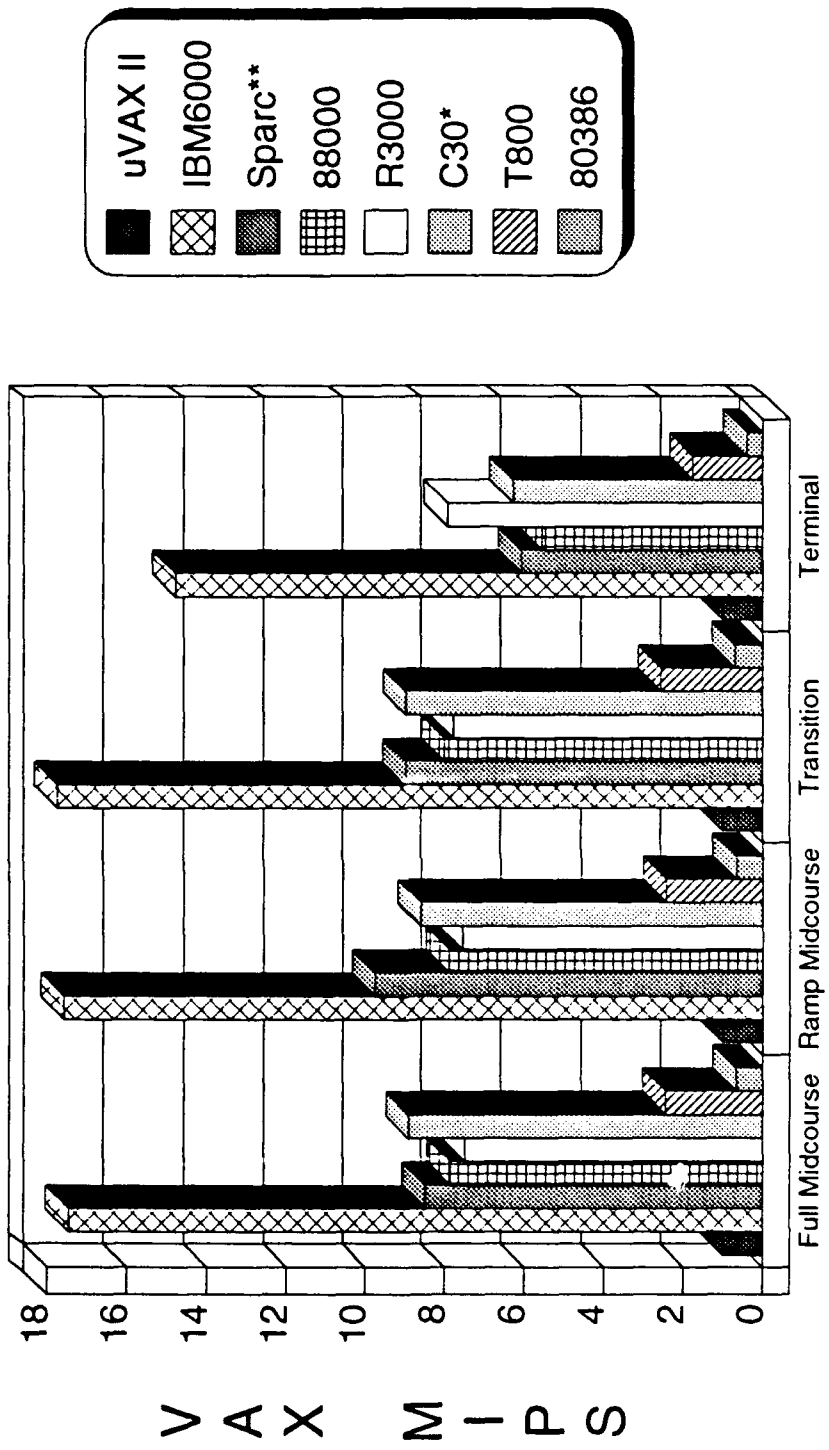


Figure 5. OGLI Benchmarks at 20 MHz

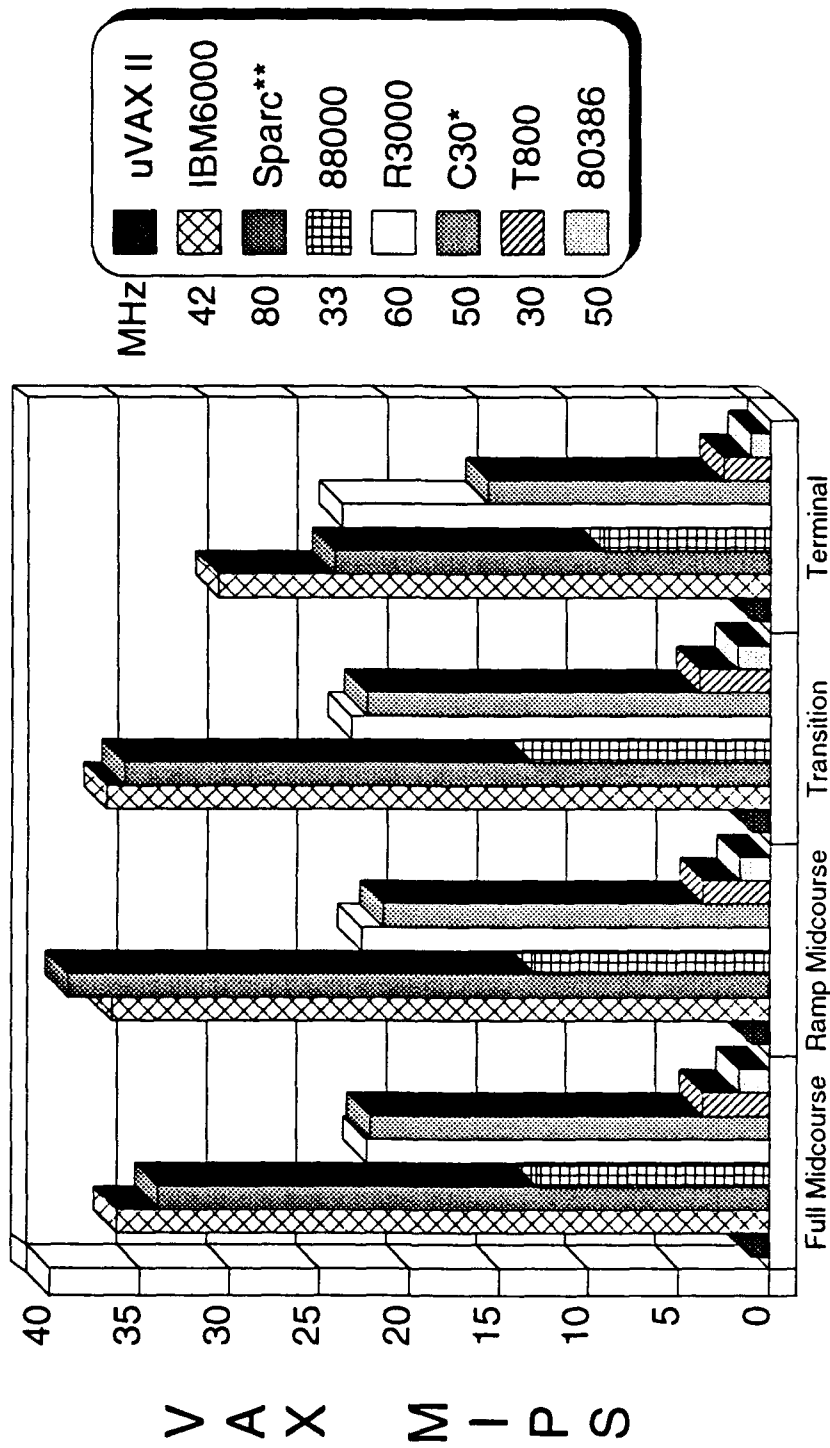


Figure 6. OGLI Benchmarks at Max Clock

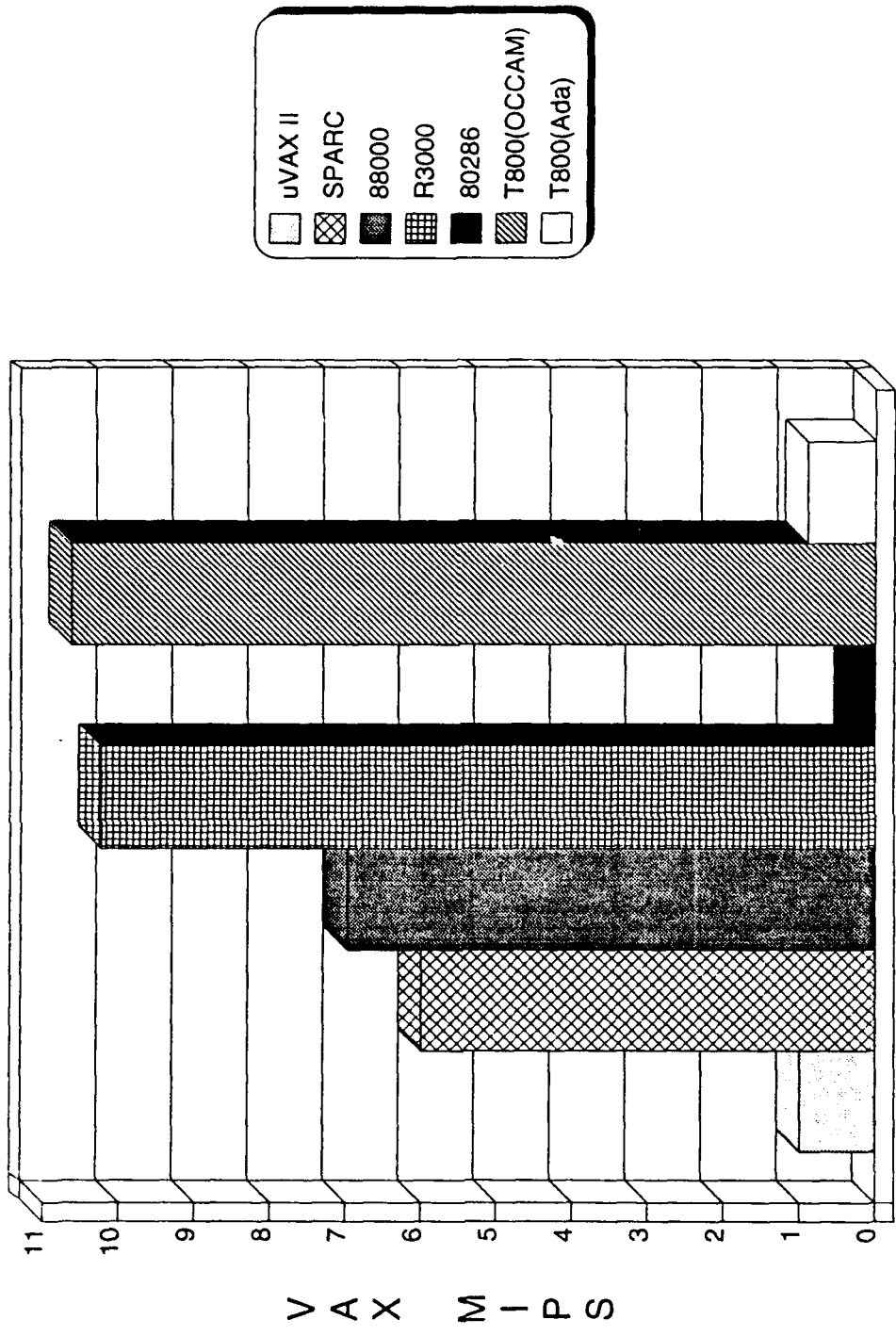


Figure 7. Step 1 Benchmark at 20 MHz

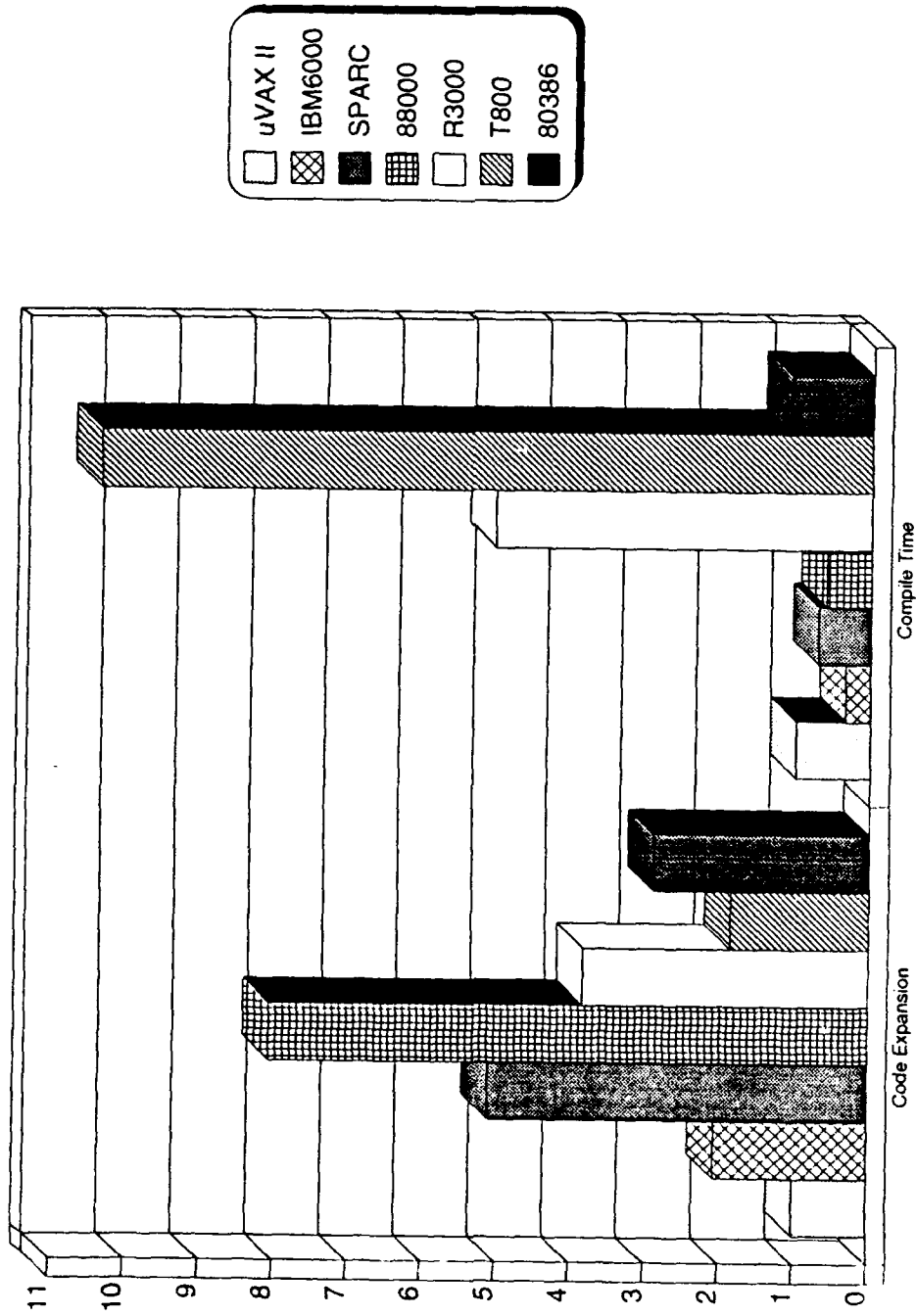


Figure 8. Code Expansion and Compile Time

a. Execution Speed (VAX-MIPS)

Table 1 and Figures 4 through 6 show the execution speed results for the four OGLI benchmarks: Full_Midcourse, Ramping_Midcourse, Transition, and Terminal. The speeds are presented in four forms: measured execution time, speed relative to the MicroVAX II (VAX-MIPS), relative speed normalized to 20 MHz, and relative speed normalized to the maximum clock. Section II.1c discussed the merits, drawbacks, and philosophy of these measurements.

The bottom of Table 1 shows the average results for the four benchmarks. Averaging the four together is not statistically valid but is presented for a rough estimate.

As for actual execution time, the goal was to be less than 10,000 microseconds. All the RISC processors easily exceeded this goal, along with the C30 signal processor. The T800 also met the goal but with a closer margin. The CISC 80386 and MicroVAX II were too slow to meet the goal. Even when the results for the two 80386 systems are normalized for their maximum clock (50 MHz), they were too slow.

For the more lenient requirements of the current OGLI system (that is, less than 100,000 microseconds for the midcourse phases and less than 20,000 microseconds for the terminal phase), all RISC processors and the 40 MHz 80386 system are fast enough, while the 25 MHz 80386 system is too slow. Note the different results obtained for two 80386 systems, using the same Ada compiler. Even when the results are normalized for 20 MHz, the 40-MHz system is around six times more efficient than the 25-MHz system. This implies that some system level feature is slowing down the 25-MHz system. This highlights the "apples and oranges" and scalability problem with benchmarking.

Figure 5 shows the results normalized to a common clock speed of 20 MHz. The IBM System 6000 performance far exceeded all processors, doubling the speed of most RISC processors. This speed is probably due to the internal parallelism of instructions that were used extensively for this application. The other three RISC processors (SPARC, 88000, R3000) had similar results. The SPARC processor speed slightly exceeded the other two. The C30 signal processor surprisingly handled the general purpose Ada code well (as opposed to mathematical code specifically turned for its architecture). The C30 processed at the same level the RISC processors. The transputer fell surprisingly behind the others probably due to its inefficient new compiler. When the Ada compiler matures to the level of the transputer OCCAM compiler, the transputer is expected to perform similarly to the RISC processors. Results for the transputer comparing Ada to OCCAM are discussed later. Finally, the CISC processors greatly lagged the RISC processors.

The pattern for the first three benchmarks (Full_Midcourse, Ramp_Midcourse, and Transition) are practically identical, indicating similar type of processing. The fourth benchmark (Terminal) shows the difference one benchmark can make. The R3000 was much more efficient with

this benchmark than the others, while the other RISC processors were still grouped together. Note that the R3000 actually processed at the same speed relative to the VAX (VAX-MIPS) for the four benchmarks, while all the other processors dropped in relative speed to the VAX for the terminal benchmark.

Figure 6 shows the results extrapolated to the speed normalized to the maximum clock for each processor. The microprocessors that have versions manufactured in ECL technology obviously have an advantage. However, the CMOS 41.6 MHz IBM System 6000 still is faster than the ECL 80 MHz SPARC, but they are close. The R6000, also in ECL, has the next best performance. However, the CMOS C30 at 50 MHz has almost identical performance as the R6000 at 60 MHz. Since the processor vendors keep coming out with faster versions, these results must be continually updated. Some vendors probably have faster versions developed but have not publicized this information.

Results for a small subset of OGLI (called Step 1) are shown in Table 2 and Figure 7. Step 1 was converted from Ada to OCCAM. The transputer running OCCAM was ten times more efficient than the Ada version. The transputer using OCCAM performed at the same level as the faster RISC (in this case R3000). Again, note this is just one small benchmark. For example, note how much better the R3000 performed than the other RISCs, which is not true for the overall GN&C application benchmark.

b. Compile Time

The compilation times for the processors are shown in Table 3 and Figure 8. The compilation time is the time required to compile and link the overall OGLI benchmark Ada program (which includes the four main benchmarks: Full_Midcourse, Ramp_Midcourse, Transition, and Terminal; the timing loop code; and a supporting package called global). Compilation time is not important for the actual execution of an embedded system but does give a feel for the "user-friendliness" during the design phase. If the developer has to wait a long time for compilation between changes to the program, development time can be long. Compilation time generally does not reflect the speed of the embedded processor; it reflects the efficiency of the host system.

The compilers for the RISC processors, which served as their own host (IBM System 6000, SPARC and 88000), had very quick compilation times. As with the execution time results, the IBM System 6000 was twice as fast as the SPARC and 88000 compilers. The compilers for the CISC processors, which served as their own host (MicroVAX II and 80386), had reasonable compilation times. However, the compilers hosted on a CISC systems, targeted to RISC processors (R3000 and T800) had excessively long compile times. Since they are embedded processors with remote hosts, they do not have an operating system already at their disposal. Consequently the run-time environment is also being linked in, which requires more time. An hour and 20-minute wait between compilations can be frustrating. To minimize this time during development, the suggestion is that instead of compiling all codes at once, design the code such that it is broken up into

manageable packages and procedures. In that case, only the portion of code being worked on would have to be recompiled.

c. Code Expansion

Table 3 and Figure 8 also show the code expansion for each processor. This is the size in bytes of the executable OGLI benchmarks (including the timing loop and all four benchmarks) and the size relative to the MicroVAX II. One of the disadvantages of RISC processors is that they generally need to use more instructions to get the same work done as the CISC complex instructions. Therefore, they require a higher memory for the embedded system and mechanisms (like fast cache memory) to quickly move all these instructions in order to keep the processor pipeline full.

The results for the OGLI benchmarks reinforce this observation. The CISC MicroVAX II had the smallest code size. Most of the RISC processors (SPARC, 88000, R3000) had large code expansion. The 88000 RISC had the worst code expansion, requiring eight times more memory than the MicroVAX II. The 80386 and T800 were in the middle. However, the IBM System 6000 compiler had the second smallest code expansion, which is surprising for a RISC architecture. This implies that either the IBM Ada/6000 compiler produces very efficient code or the IBM System 6000 architecture is more like the CISC.

SECTION III

PARALLEL IMPLEMENTATION OF OGLI

In an effort to improve the processing performance of the OGLI code, an experiment was made using parallel processing. In this section, the transputer hardware is discussed, followed by the parallel processing approach, and finally the benchmark results are presented.

1. TRANSPUTER BACKGROUND

The transputer was designed specifically with parallel processing in mind. Each transputer has four pairs of high speed serial I/O ports (T-links). The transputer family of processors were designed for low cost, so that massively parallel systems can be affordable. On each transputer T800 is 4K of internal RAM, a 32-bit processor, a 64-bit floating point unit, and timers.

To aid in the development of transputer systems, INMOS packages the transputers in various modules that can easily be plugged into a "motherboard," which in turn can be plugged into a personal computer bus. In this case, we have the IMS B008 motherboard, which has up to ten slots for modules. The primary module, which communicates directly with the personal computer, needs more memory to handle the Ada run-time. This module has a T800 running at 25 MHz and 4 Mbytes of RAM external to the T800, but dedicated to the T800. The other seven modules have 20 MHz T800s and 1 Mbyte of external memory each.

On the motherboard, each module has two hardwired serial I/O connections (hard links) to each of its neighbors. This leaves two remaining pairs of serial input/output for each processor (soft links) that can be routed to any other open pair via software changes. This setup gives the designer much flexibility in changing the hardware configuration of the processors without much effort. Therefore, there are a maximum of seven processors available to configure for this parallel processing experiment.

For this architecture, each processor has its own memory. There is no shared or blackboard memory; therefore, the transputers must communicate all values to each other when needed.

2. APPROACH

Today, there are many hardware systems designed efficiently for parallel processing. The real difficulty is not designing parallel hardware, but writing parallel software for an application that optimally takes advantage of the parallel hardware. Algorithms and systems designed by engineers are generally created by a serial thought process. Humans have been trained to think serially.

Therefore, the first step is to convert the serial algorithm into a parallel version. The code must be partitioned into parallel subroutines. The timing of the subroutines must be analyzed, and the communication throughput between the subroutines must be calculated. From this information, the appropriate level of granularity can be determined.

a. Partition

The first step is to partition the code into smaller pieces, which can be implemented in parallel. For the OGLI benchmark code, the guidance law naturally had 25 steps. Each formed a partition. Some of these partitions were bigger than others or did multiple functions, and thus, could be subpartitioned. The level to which the code is partitioned is referred to as granularity.

A dependency analysis is done between partitions. This action determines which partitions depend on variables passed from other partitions that in turn determines which partitions can be implemented parallel to each other and which must be done serially. Figure 9 shows the dependencies of the original 25 OGLI partitions. Some were dropped or absorbed. Note the parallelism at the widest point only includes four parallel partitions and several serial partitions at the end. The smaller boxes inside the partition boxes represent code that is independent of each other, so could be subpartitioned if necessary. With further analysis, some code may be moved from one partition to another. This would be beneficial if it eliminated a dependency and allowed the partitions to be in executed in parallel instead of serially.

b. Timing Analysis

The next step is to calculate the time required to execute each partition and subpartition. For the OGLI partitions, each was executed as a benchmark on a T800 to get the timing. These times are inside each partition in Figure 9 in microseconds. If possible, a worst and best case time were determined.

Given this timing and dependency information, the use and idleness of the processors can be determined. For the OGLI partitions in Figure 9, it appears that two processors can be kept busy most of the time. Even though there are extra partitions that can be implemented on two extra processors in parallel, they will be idle most of the time. These partitions can be combined and executed on one processor and still not slow down the overall system.

c. Throughput/Communication Analysis

The next step is to study the communication required between the partitions and subpartitions. The communication overhead between processors is usually the limiting factor in parallel processing. Even though an algorithm can be decomposed into finer and finer granularity, if the communication time between processors is greater than the execution

time for the function on the processor, this level of parallel processing will provide no benefit. In this situation, adding additional processors will actually slow down the processing.

For OGLI, the communication time is related to the number of variables passed. The communication time between processors was calculated for the transputer system. It was found, on average, to be around 77 microseconds for two-way communication and 38 microseconds for one-way. Given this information, any partition that can be executed in less time would have too much communication overhead to be run on a separate processor and could be executed quicker on the processor that needs the information.

Using the partitions, dependency analysis, execution timing, and communication overhead information, the partitions were mapped to the transputers as shown in Figure 10. The arrows note when communication must be done between processors. Some of the partitions were subpartitioned (that is, steps 1a, 1b, and 1c).

3. RESULTS

The results for the parallel implementation of OGLI on three T800 transputers is shown in Table 4. Given an ideal parallel algorithm, which can keep all processors busy and no communication overhead, the speed improvement for three processors over a single processor would be 200 percent. For the OGLI algorithm, over half of the processing is done sequentially. Therefore, the 76 percent speed improvement was expected. Adding an additional transputer would not increase the speed since the communication overhead would overcome the benefits of any further parallel execution. Reducing the number of transputers from three to two reduced the speed improvement to around 49 percent.

It involves extensive the time and trouble to optimize a parallel system. Although performance gains are achieved, greater gains may be made through simpler methods--for example, upgrading the processor to a faster clock speed or using a more efficient version of an Ada compiler. The additional processors also add cost, weight, and space to an embedded system. This guidance law can be implemented on a single processor with plenty of margin.

However, a parallel transputer architecture would still be beneficial for an overall system as in Figure 1. For example, the guidance code could be executed on one processor, the seeker on a couple of processors, and the IMU code on another processor.

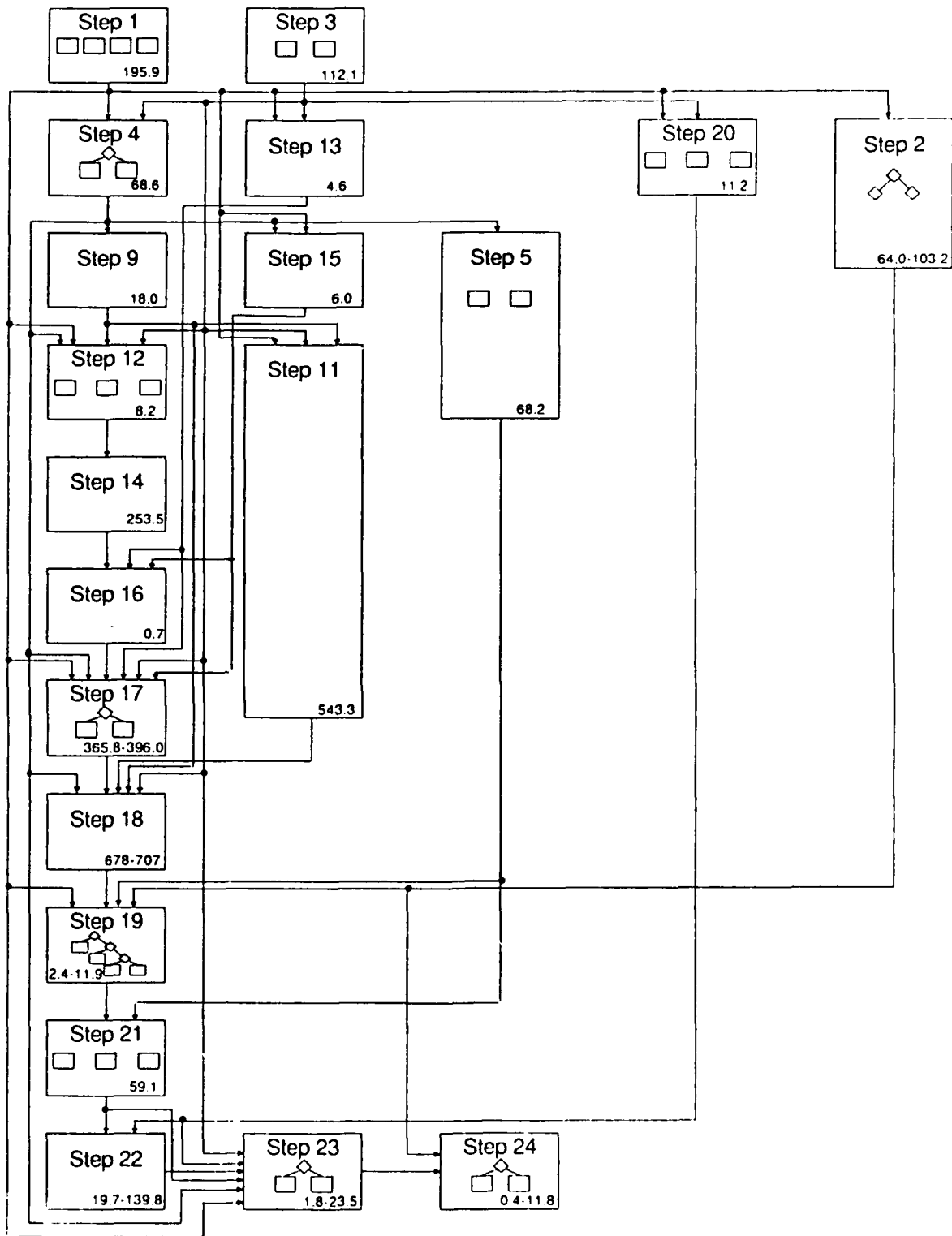


Figure 9. Partition and Dependency Analysis

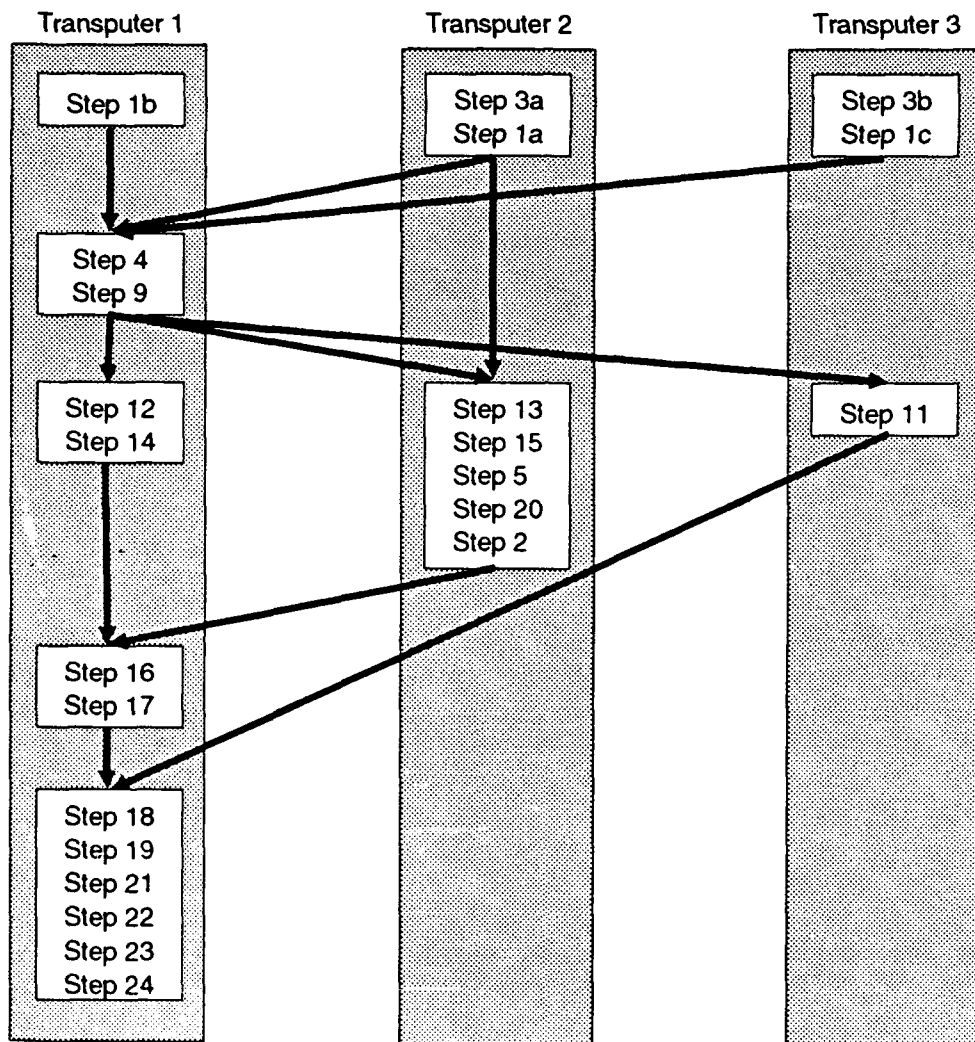


Figure 10. OGLI Processing Flow on Three Transputers

TABLE 4. OGLI BENCHMARKS ON TRANSPUTERS

Benchmark	Number of Processors	Exec. Time (usec)	Speed Relative to uVAX II	Percentage of Speed Increase over One
Full_Midcourse	1	8986.3	3.07	--
	3	5118.3	5.39	76
Ramping_Midcourse	1	8950.2	3.05	--
	3	5097.3	5.36	76
Transition	1	8087.8	3.21	--
	3	4708.9	5.51	72
Terminal	1	2173.0	2.20	--
	3	1208.6	3.96	80

SECTION IV

CONCLUSION AND RECOMMENDATIONS

Overall, the RISC processors far out-performed the CISC processors for our guidance law application. The IBM System 6000 was the best performer, demonstrating the power of its internal parallel instructions. However, the IBM System 6000 architecture is currently a dedicated workstation, while the other RISC processors are more suited for embedded applications. IBM is working on an embedded version of the 6000.

All the RISC processors met our processing requirements by a large margin. Surprisingly, the TI C30 signal processor also performed at the same level as the RISC processors. The TI C30 is the first signal processor to have an Ada compiler. The transputer parallel processor did not perform as well as the RISC processors (when a single transputer was used), but did meet the processing requirements. As the Ada compiler of the transputer matures and becomes more efficient, the transputer is expected to perform closer to the RISC levels.

The CISC processors executed too slowly to meet our requirements. However, the CISC processors did offer the advantage of little code expansion (small memory size) compared to the RISC processors. The IBM System 6000 was the exception of the RISC processors, having a small code expansion similar to the CISC processors. Having both a quick execution time and a small code expansion indicates an efficient compiler.

When executing the guidance law in parallel, three transputers offered the most efficient execution. Three transputers increased performance 76 percent compared to a single transputer. Adding more transputers actually decreased performance due to communication overhead. The three transputers in parallel were able to process at or about the same level as the RISC processors.

Again, these benchmark results only apply to this specific application. Given different benchmarks, completely different results could occur. These results are really just a snapshot in time. To remain current, we must continually update benchmarking as new processors are introduced, as new Ada compilers are released, and as the benchmarking algorithm changes.

REFERENCES

1. W. Brauckmann, Guidance Instruction Set Architecture (GISA) Phase II, AFATL-TR-90-58, Westinghouse Electric Corp., Baltimore, Maryland, October 1990.
2. S. Cohen and T. Taylor, Common Ada Missile Packages - Phase 2 (CAMP-2), Volume III. CAMP Armonics Benchmarks, AFATL-TR-88-62, McDonnell Douglas Astronautics Company, St Louis, Missouri, November 1988.
3. T. Leavitt and K. Terrell, Ada Compiler Evaluation Capability (ACEC) Reader's Guide, AFWAL-TR-88-1094, Boeing Military Airplane, Wichita, Kansas, August 1988.
4. T. Leavitt and K. Terrell, ACEC Technical Operating Report: User's Guide, AFWAL-TR-88-1094, Boeing Military Airplane, Wichita, Kansas, August 1988.
5. M. E. Sisle, F. Zupancic, and D. Luciano, Optimal Guidance Law Implementation, AFATL-TR-88-103, Raytheon Company, Bedford, Massachusetts, October 1988.
6. Verdix Corporation, VADS Verdix Ada Development System, Version 6.0, User's Guide, Sun-4 SunOS, 5 April 1989.
7. TeleSoft, Telegen2 for UNIX/88k, Host User's Guide, 25 May 1990.
8. IBM, AIX Ada/6000, Release 2.0, User's Guide, March, 1991.

BIBLIOGRAPHY

Advanced Technology Group, "RISC Enough for the Next Generation," Computer Design, November 13, 1989.

Alsys, Alsys Ad Ada Compilation System Environment, Version 4.4, Burlington, Massachusetts, March 30, 1990.

Brauckmann, W., Guidance Instruction Set Architecture (GISA) Phase II, AFATL-TR-90-58, Westinghouse Electric Corp., Baltimore, Maryland, October 1990.

Cohen, S., and Taylor, T., Common Ada Missile Packages - Phase 2 (CAMP-2), Volume III. CAMP Armonics Benchmarks, AFATL-TR-88-62, McDonnell Douglas Astronautics Company, St Louis, Missouri, November 1988.

Gross, T., Core Assembly Language Instruction Interface for RISC-style Microprocessors, Version 3.3, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 1989.

IBM, AIX Ada/6000, Release 2.0, User's Guide, March, 1991.

IBM, "IBM RISC System/6000 Processor," IBM Journal of Research and Development, Volume 34, Number 1, January, 1990.

INMOS, Transputer Reference Manual, Prentice Hall, New York, 1988.

Intel, 386 DX Programmer's Reference Manual, 1990.

Intel, 80960 Programmer's Reference Manual, Santa Clara, California, 1988.

InterACT, InterACT Ada Mips Cross-Compiler System, Release 1.0, October 31, 1989.

Johnson, T. L., "A Comparison of MC68000 Family Processors," Byte, September 1986.

Kane, G., MIPS RISC Architecture, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1988.

Leavitt, T., and Terrell K., ACEC Technical Operating Report: UseUser's Guide, AFWAL-TR-88-1094, Boeing Military Airplane, Wichita, Kansas, August 1988.

Leavitt, T., and Terrell K., Ada Compiler Evaluation Capability (ACEC) Reader's Guide, AFWAL-TR-88-1094, Boeing Military Airplane, Wichita, Kansas, August 1988.

Motorola, MC88100 RISC Microprocessor User's Manual, Second Edition, Prentice Hall, New Jersey, 1990.

BIBLIOGRAPHY (Concluded)

Sisle, M. M. E., Zupancic, F., and Luciano, D., Optimal Guidance Law Implementation, AFATL-TR-88-103, Raytheon Company, Bedford, Massachusetts, October 1989.

TeleSoft, Telegen2 for UNIX/88k. Host User's Guide, 25 May 1990.

Texas Instruments, TMS320C3x User's Guide, 1990.

Verdix Corporation, VADS Verdix Ada Development System, Version 6.0, User's Guide, Sun-4 SunOS, 5 April 1989.

Wilson, R., "Motorola 68040 Challenges RISC Microprocessors Head On," Computer Design, February 1, 1990.

DISTRIBUTION LIST
(WL-TR-92-7015)

DTIC/DDAC Cameron Station Alexandria VA 22304-6145	2	Eglin offices:	
AUL/LSE Maxwell AFB AL 36112-5564	1	WL/MNOI (Scientific and Tech. Info. Facility) WL/CA-N	1 1
HQ USAFE/INATW APO NY 09012-5001	1	WL/FIES/SURVIAC Wright-Patterson AFB OH 45433-6553	1
AFSAA/SAI Washington DC 20330-5420	1		

Eglin offices:

WL/MNPX	1	ASC/XRH	1
WL/MNAV	1	Wright-Patterson AFB OH 45433-6503	
WL/MNM	1		
WL/MNAG	2	ASC/ENSTA	1
WL/MNSI	1	Wright-Patterson AFB OH 45433-6503	
ASC/XRC	1		
AFDTC/PA	1	AFIA/INT	1
		Bolling AFB DC 20332-5000	
WL/TXA	1		
Wright-Patterson AFB OH 45433-6523		EOARD/LDV	1
		Box 14	
		FPO NY 09510-0200	
Wright-Patterson AFB OH 45433-6553:			
		Commander	1
WL/FIM	1	Naval Weapons Center (Code 3431)	
WL/CA-F	1	Attn: Technical Library	
WL/FIB	1	China Lake CA 93555-6001	
WL/FIBA	1		
WL/FIGX	1	NASA Langley Research Center	1
WL/FIGC	1	Technical Library - MS 185	
		Attn: Document Cataloging	
		Hampton VA 23665-5225	
Commander	1		
U. S. Army Missile Command			
Redstone Sci Info Center			
Attn: AMSMI-RD-CS-R/Documents			
Redstone Arsenal AL 35898-5241			