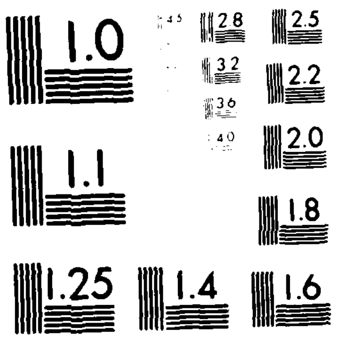
AD-A169 981 HIGH PERFORMANCE PARALLEL COMPUTING(U) TEXAS UNIV AT AUSTIN DEPT OF COMPUTER SCIENCES J C BROWNE ET AL. 22 JAN 86 AFOSR-TR-86-0457 F49620-84-C-0020 1/1

UNCLASSIFIED F/G 9/2 NL


1.0
4.5
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

AD-A169 981

2

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 86-0457
6a. NAME OF PERFORMING ORGANIZATION The University of Texas at Austin	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research	
6c. ADDRESS (City, State and ZIP Code) Department of Computer Sciences Austin, Texas 78712	7b. ADDRESS (City, State and ZIP Code) Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR F49620-84-C-0020	
8c. ADDRESS (City, State and ZIP Code) Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A3	

11. TITLE (Include Security Classification) HIGH PERFORMANCE PARALLEL COMPUTING

12. PERSONAL AUTHOR(S) J. C. Browne and G. J. Lipovski

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM 2/1/84 TO 1/31/85	1986, Jan. 22	20 (plus papers)

16. SUPPLEMENTARY NOTATION

DTIC ELECTE JUL 2 3 1986 S D

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR	

DTIC FILE COPY

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The 1984/85 accomplishments of the research project "High Performance Parallel Computing" included bringing the prototype of the Texas Reconfigurable Array Computer (TRAC) to a configuration and to a state of stability where it could support execution of simple assembly language programs, initial development of a unified model of parallel computation which is a basis for a programming environment uniting process and data flow models of parallel computation, bringing to operational status on an alternative host one of the two parallel programming languages (the Computation Structures Language, CSL) originally intended for use on TRAC, exploration of the expressive capabilities of this programming language, initiation of development of a graphical programming language based on the unified model of parallel computation mentioned preceding, major progress on a graphically interfaced Petri Net-based performance modeling system for parallel computations and development of algorithms for scheduling of circuits to realize configurations in configurable banyan network based computer architectures.

(continued on separate sheet)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Capt. A. L. Bellamy	22b. TELEPHONE NUMBER (Include Area Code) (202) 767-
22c. OFFICE SYMBOL NM	

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

Abstract (continued)

The TRAC configuration which became available for use during this period is a four-processor nine-memory system with two 10 MBYTE Winchester disks. The VAX 11/750 obtained under the DoD Research Equipment program is used as both front- and back-end for TRAC. This configuration is not able to support development of major software systems but can be used for small scale experiments in reconfigurable computation. Several such experiments were conducted including one which attached an image input device to the auxiliary resources interface developed in 1983 and reported in the Final Report for Contract F49620-83-C-0049.

Substantial existing Fortran programs, up to 4,000 lines of code, have been given parallel structure in the Computation Structures Language (CSL) and used as test programs for the concepts and implementation of CSL.

The computations of the scheduling algorithms developed for selection of circuits to realize configurations in banyan network architectures are distributed among the nodes of the switch. This distribution and resulting parallel execution gives log(n) execution time for circuit scheduling. (n is number of base or apex nodes in the network.)

It was, during this year, recognized that the network structure which couples processors and memories in TRAC can be adapted and extended to couple computers to external memory systems to provide a very high performance highly parallel I/O system.

AFOSR-TR. 86-0457

Final Report to the Air Force Office of Scientific Research

On Contract Number F49620-84-C-0020

High Performance Parallel Computing

from
J. C. Browne
Department of Computer Sciences
G. J. Lipovski
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712

20 January 1986

86 7 23 156

ABSTRACT

The 1984/85 accomplishments of the research project "High Performance Parallel Computing" included bringing the prototype of the Texas Reconfigurable Array Computer (TRAC) to a configuration and to a state of stability where it could support execution of simple assembly language programs, initial development of a unified model of parallel computation which is a basis for a programming environment uniting process and data flow models of parallel computation, bringing to operational status on an alternative host one of the two parallel programming languages (the Computation Structures Language, CSL) originally intended for use on TRAC, exploration of the expressive capabilities of this programming language, initiation of development of a graphical programming language based on the unified model of parallel computation mentioned preceding, major progress on a graphically interfaced Petri Net-based performance modeling system for parallel computations and development of algorithms for scheduling of circuits to realize configurations in configurable banyan network based computer architectures.

The TRAC configuration which became available for use during this period is a four-processor nine-memory system with two 10 MBYTE Winchester disks. The VAX 11/750 obtained under the DoD Research Equipment program is used as both front- and back-end for TRAC. This configuration is not able to support development of major software systems but can be used for small scale experiments in reconfigurable computation. Several such experiments were conducted including one which attached an image input device to the auxiliary resource interface developed in 1983 and reported in the Final Report for Contract F49620-83-C-0049.

Substantial existing Fortran programs, up to 4,000 lines of code, have been given parallel structure in the Computation Structures Language (CSl) and used as test programs for the concepts and implementation of CSL.

The computations of the scheduling algorithms developed for selection of circuits to realize configurations in banyan network architectures are distributed among the nodes

of the switch. This distribution and resulting parallel execution gives log(n) execution time for circuit scheduling. (n is number of base or apex nodes in the network.)

It was, during this year, recognized that the network structure which couples processors and memories in TRAC can be adapted and extended to couple computers to external memory systems to provide a very high performance highly parallel I/O system.

1. Research Objectives

The research objectives of the project "High Performance Parallel Computing" was an integrated approach to parallel computation coupling the development of a novel reconfigurable parallel computer architecture, the Texas Reconfigurable Array Computer (TRAC), with the development of software for the architecture and design of algorithms which could effectively utilize the capabilities of the architecture. This project, which we shall refer to as the TRAC Project, was initiated in 1978. The research accomplishments reported for 1984/85 are in context of this substantial and long-lived project. The specific objectives for 1984/85 included bringing the prototype of TRAC to a configuration and to a state of stability where it could support development of software systems and applications, to bring into experimental use the software systems designed and partially implemented in previous research and to explore the effectiveness of the reconfigurable execution environment provided by TRAC on significant algorithms.

A summary of architectural concepts for TRAC will provide a context for the reporting of the research accomplishments given following. The fundamental concept of TRAC is that an effective execution environment for parallel computations can be realized by having a computer architecture which can be configured to the arithmetic and communication requirements of the algorithms and applications it is executing. Configurability is attained by having processor and memory resources connected by a banyan interconnection network which supports the establishment of circuits at runtime and which also implements memory to processor packet routing. The processors are placed at the apex of the network and the memories at the base of the network. Figure 1 illustrates this concept with a four-processor nine-memory configuration connected by a switch with nodes having spread of two and fan-out of three. A "computer" is realized by establishing circuits in the network coupling processors to memories and coupling "Computers" to "Computers." Figure 2 shows a configuration realizing an MIMD configuration with each "computer" being made up of two processors coupled through the switch, and two memory units. The two computers also share a memory unit. It is straightforward to see that this procedure can be used to construct a spectrum of architectures ranging across SIMD and MIMD and including both message-based and shared-memory-based configurations. A more extensive discussion can be found in Browne and Lipovski [BRO82a].

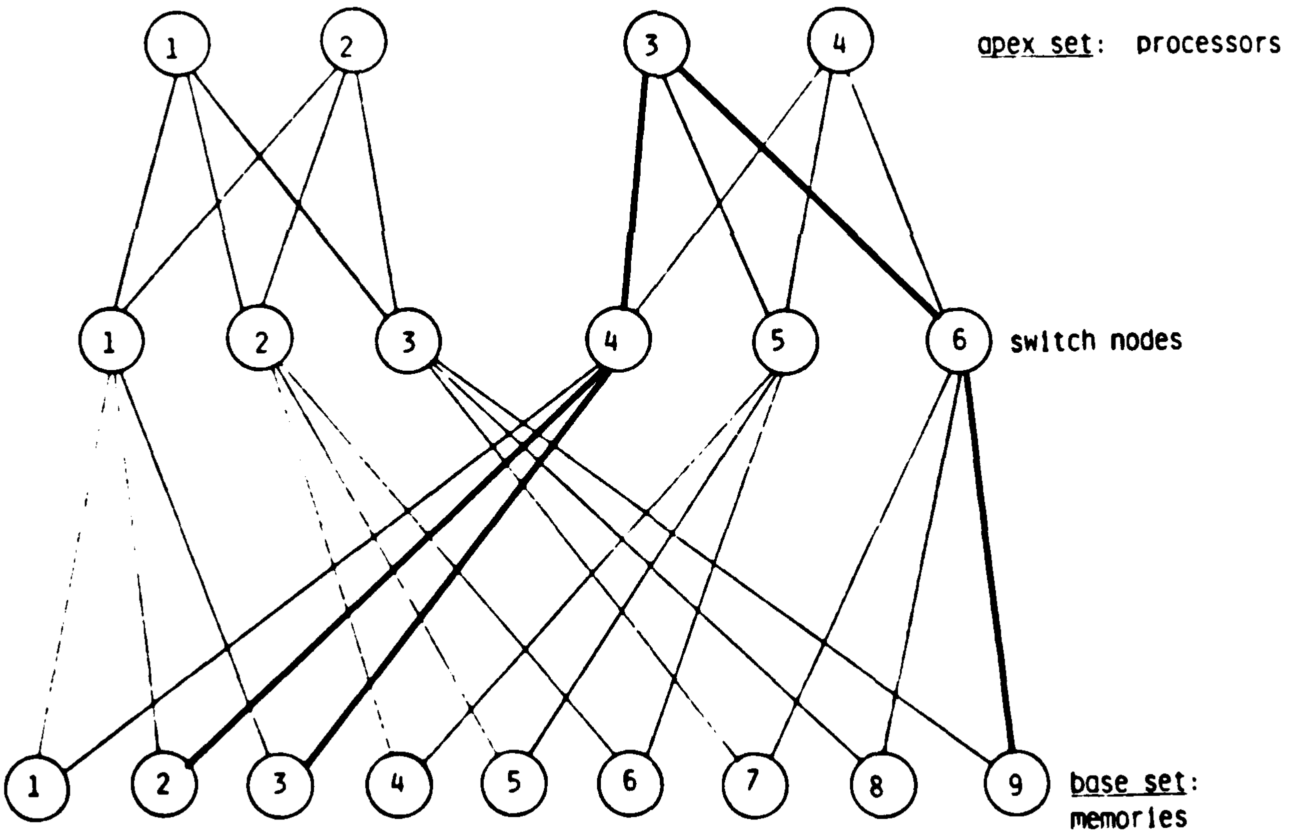


Figure 1

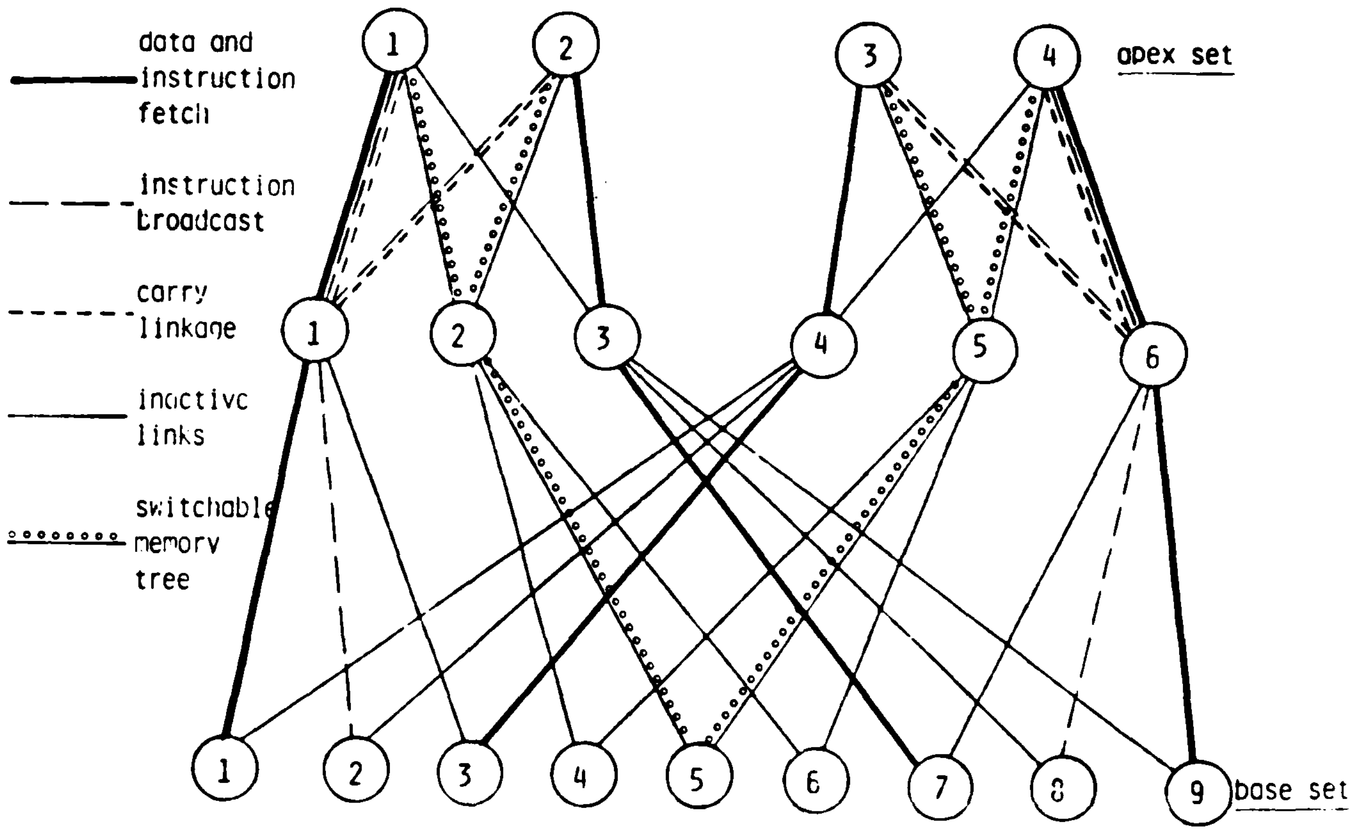


Figure 2

2. Research Accomplishments

All of the objectives were attained in greater or lesser degree. The subsections which follow give the accomplishments in summary form relating them to published papers where possible.

2.1. Hardware Design and Development

The major accomplishments for 1984/85 in the area of hardware design and development were redesign of the memory board, coupling of the VAX 11/750 obtained under a DoD Research Equipment Grant (Grant Number AFOSR-83-0315) to TRAC to serve as a front- and back-end and bringing the four-processor nine-memory configuration to a level of stability where it can serve as an execution environment for simple programs.

The memory board which was actually carried to completion for inclusion was a redesign of the original board which corrected timing errors and incorporated much greater margins for stability. This redesign was carried out on the VALID Logic ECAD design system purchased with the matching funds provided by the University of Texas in connection with DoD Research Equipment Grant Number AFOSR-83-0315. The addition of this more stable memory board enabled a sustained push of hardware testing (which was made possible by the Tektronix Logic Analyzer purchased under the DoD Research Equipment Grant Number AFOSR-83-0315) which brought the four-processor nine-memory configuration of TRAC to a point where it could be and was used for execution of demonstration codes and simple applications. The assembler and loader which had been hosted on the Research DEC 2060 of the Computer Science Department were ported to the TRAC VAX to facilitate code development and loading. The most significant application attempted was coupling of an image digitizer to the Auxiliary Resource Interface (ARI) referenced in the Final Report for Grant F49620-83-C-0049. Code for simple image processing applications was developed and run on the digitized images.

2.2. Algorithms and Theory

A fundamental problem for a configurable network architectured computer architecture is development of algorithms for selection of the processors, memories and switch resources with which to establish specific computer configurations. The problem arises because the Banyan interconnection network is a blocking network which cannot simultaneously realize circuits giving full connectivity between all processors and all memories. Effective use of the processor and memory resources thus requires effective algorithms for construction of configurations. In addition to yielding efficient use of resources, the algorithm must be sufficiently efficient to be frequently executed during the course of execution of a computation. It must be applicable to partial configurations of resources since the total system resources may be shared by several jobs. Feo [FEO85] has developed and evaluated by simulation a spectrum of algorithms. All have the common property that the computations of the algorithm are distributed across the nodes of the network. All function by broadcast of signals through the network, first from apex nodes to base nodes and then back to the apex nodes. Each pass generates state information about the number of paths through the network which will be consumed by selection on a given resource configuration. There exists an algorithm which can be shown to minimize loss of connectivity in the network [BIT84]. This algorithm is rather complex. The results of Feo's research show that simple algorithms whose functionality can be implemented by adding fewer than 150 gates to the switch nodes give near optimal results over a wide range of network states with log(n) computation time where n is proportional to the number of apex and base nodes.

There are a great number of models of parallel computation. Some of the more popular are the process model, the several flavors of data flow and the functional/applicative model. It has been the case in the past that each of these models of computation has spawned its own programming languages and architectures. Browne [BRO85a, BRO85b] has developed a unified model of parallel computation which integrates all of the common models of parallel computation at a somewhat abstract level. The essential concept is to generalize dependency relations and to formulate computations as graph structures where the nodes are schedulable units of computation and the arcs are generalized dependency relations. This model of parallel computation allows a single computation graph to include several different types of dependency relations. This model of parallel computation leads naturally to the concept of a graphical programming language. This paradigm for graphical programming is described briefly in the following section.

2.3. Parallel Programming System

Implementation of the Computation Structures Language (CSL) [BRO82b] on the dual CDC Cyber 170/750 configuration of the University of Texas Computation Center was completed during 1984/85. The choice of the dual Cybers as the host for implementation of CSL was made because the configuration of TRAC which was available would not support the runtime system of CSL. The four-processor nine-memory configuration has a total of 576 KBYTES of memory. It did not have the interrupt handling capabilities implemented. There remained problems of intermittent hardware failures. The circumstances made it impractical to attempt implementation of the operating system and programming language systems designed for TRAC. The Computation Center of the University of Texas added to the UT-2D operating system the necessary primitives for implementation of generalized dependency relations. A number of programs of existing Fortran programs of substantial size have been given parallel structure at the module level and executed under control of the CSL runtime system. The largest of these programs is a 4,000 line molecular integrals code.

The CSL runtime system was also used as a vehicle for the study of parallel structuring of resource management systems [ODE85]. The result was a relationship defining the degree of parallelism necessary for the CSL runtime system to execute a given workload.

We have also used CSL as a vehicle for study of the amount of parallelism in an algorithm which can be readily captured in programming languages. CSL is a language for representing dependency graphs at the task level and for expressing a traversal of the graph to execute the computation. CSL incorporates both message and shared memory models for implementation of dependency relations. It was found in several studies that the use of both types of dependency relations aided in obtaining compact parallel computation structures which yield most of the potential parallelism in an algorithm. Appendix A develops one example CSL program from a specification of an algorithm for solution of a set of linear equations with a triangular coefficient matrix.

We also initiated during this period implementation of two other programming systems for parallel computation. The Task Level Data Flow Language (TDFL) uses an explicit specification of a dependency graph in terms of a subset of the generalized dependency relations defined by Browne [BRO85b] but leaves the traversal of the graph to be determined dynamically at runtime. A prototype implementation of TDFL has been accomplished since the conclusion of the period of this grant. We also have begun design and implementation of a graphical programming interface in which a programmer specifies a parallel program directly as a dependency graph where the nodes of the graph are pre-programmed schedulable units of computation and the properties of the generalized dependency graph are declaratively specified.

2.4. Performance Modeling of Parallel Computations

There are always many possible parallel structures for a given computation. The factors which can be varied include granularity of the schedulable units of computation and choice of types of dependency relation. It would be laborious to systematically explore the parameter space for complex computation structures with explicit programming. Adiga [ADI86] has been developing a performance modeling system for parallel computation structures. The conceptual basis of the modeling system is an extended form of Petri Nets. The extended Petri Nets are expressed in terms of vector replacement systems, a generalization of vector addition systems developed by Kapur [KAP82]. The performance modeling system uses a graphical interface for model specification. The modeling system has been implemented to have constructs which are direct equivalents of CSL programs. A CSL program can be directly mapped to an extended Petri Net model and vice versa. A critical aspect of any performance modeling system is validation. The ability to transform back and forth from CSL facilitates validation. A study of the design space for parallel computations is executed by writing a CSL program for one parallel structure and measuring its execution properties. The CSL program determines the structure of the model. The measurement data can then be used to parameterize and validate the extended Petri Net performance model. The model can then be used to search for "optimal" parallel structures with increased confidence. The design of this performance modeling system was done under sponsorship of this grant.

2.5. Parallel I/O Architectures

The network architecture concepts of TRAC have been extended and adapted to generate a highly parallel I/O architecture which integrates data base operations, efficient virtual memory operation and object-oriented data handling [BRO85c].

The basis of the proposed I/O architecture is to connect the external memories of the multiprocessor architecture to the computation elements via a network switch. The nodes of the switch are given the functionality to execute routing and sort/merge operations on data elements as they traverse the switch between the processors and external memories. The routing operations can be used to implement indexed store operations in the processor to memory direction and data distribution and sort/merge operations in the memory to processor direction. The external memory systems are interfaced to an object-oriented self-managing secondary memory (SMSM) [RAT84] which is in turn interfaced to the network by a specialized processor which implements data filtering operations. The SMSM's are composed of cells which can execute search operations in parallel and which also execute memory management functions such as allocation and garbage collection. The SMSM's are *short latency object-oriented caches* for the mass storage devices. Figure 3 is a schematic of this architecture.

This architecture clearly has the potential for a great deal of parallelism in the operation of I/O and data base operations. Multiple activities can be initiated at the processor level. Indexing operations are executed in parallel by the network switches. The SMSM's can execute searches in parallel and each SMSM can have internal search parallelism. Research to explore the potential of this I/O architecture was initiated under the partial sponsorship of this grant.

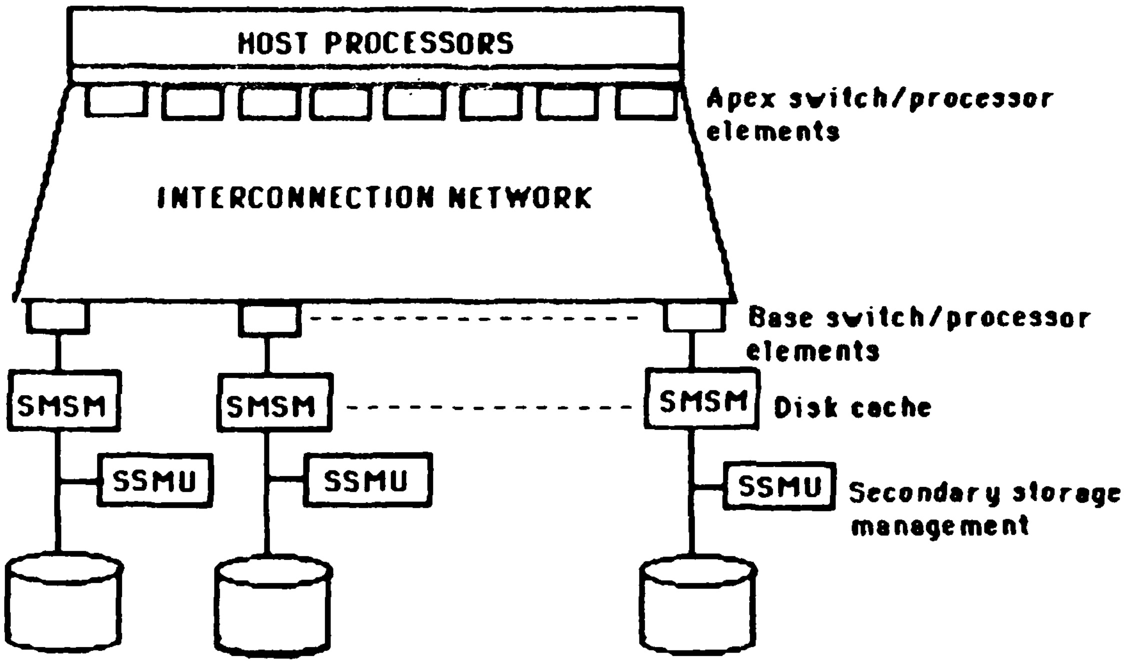

HOST PROCESSORS
Apex switch/processor elements
INTERCONNECTION NETWORK
Base switch/processor elements
SMSM
SMSM
SMSM
Disk cache
SSMU
SSMU
SSMU
Secondary storage management

Figure 3

2.6. References

The references given here are those referred to in the body of the report which are not publications reported in this Final Report. Citations not given here are to be found in the list of publications for this grant period.

[BRO82a]	Browne, J.C. and Lipovski, G.J., "Reconfigurable Network Architectured Computer Systems: An Environment for Parallel Computing," Proceedings of the International Workshop on High Level Language Computer Architecture, Ft. Lauderdale, FL, December 1982, pp. 40-49.
[BRO82a]	Browne, J.C., Tripathi, A., Fedak, S., Adiga, A. and Kapur, R., "A Language for Specification and Programming of Reconfigurable Parallel Computation Structures."
[BIT84]	Bitner, J., (Private communication).
[KAP82]	Kapur, R., "On the Synthesis and Analysis of Reconfigurable Computer Programs," Ph.D. Dissertation, Department of Electrical Engineering, The University of Texas at Austin, May 1982.
[RAT84]	Rathi, B.D., "The Design and Performance Analysis of a Self Managing Secondary Memory," Ph.D. Dissertation, Department of Electrical and Computer Engineering, The University of Texas at Austin, December 2983.

3. Papers Published

[BRO84]	Browne, J.C., "TRAC: An Environment for Parallel Computing." Proceedings of COMPCON 84, pp. 295-298.
[BRO85a]	Browne, J.C., "A Framework for Formulation and Analysis of Parallel Computation Structures," Proceedings of the 18th Hawaii International Conference on System Sciences, January 1985, p. 207.
[BRO85b]	Browne, J.C., "Formulation and Programming of Parallel Computations: A Unified View," Proceedings of the XI International Conference on Parallel Processing, Chicago, August 1985, pp. 624-631.
[FEO85]	Feo, J., Jenevein, R. and Browne, J.C., "Dynamic Distributed Resource Configurations on SW Banyans," Proceedings of the 12th Annual International Symposium on Computer Architecture, Boston,

June 1985, pp. 268-275.

[LIP85] Lipovski, G.J., Deshpande, S. and Jenevein, R., "TRAC: An Experience with a Novel Architectural Prototype," Proceedings NCC AFIPS, 1985. (Acknowledgement omitted)

3.1. Papers Submitted for Publication

[MAL86] Malek, M. and Opper, E., "The Cylindrical Banyan Multicomputer: A Reconfigurable Systolic Architecture," accepted for publication in Journal of Parallel Computing.

[BRO86] Browne, J.C., "Characterization of Parallel Computer Architectures," to appear in the Journal of Parallel Computing.

4. Personnel Associated with Project

4.1. Senior Personnel

- M. Malek, Co-Principal Investigator
- Matthew Sejnowski, Research Engineering/Scientist Associate
- Patrick Horne, Computer Systems Development Specialist
- Sanjay Deshpande, Research Engineering/Scientist Associate
- E. J. Shipsey, Research Associate

4.2. Graduate Research Assistants

- Dan Berleant
- Ashok Adiga
- John Feo
- Robert O'Dell

4.3. Undergraduate Research Assistants

- Timothy Trudeau
- Clark Burnett

5. Graduate Degrees Awarded

- Robert O'Dell, M.S., "On the Parallel Structuring of Resource Management," Department of Computer Sciences, The University of Texas at Austin, August 1985.
- Ashok Adiga, Ph.D., "Performance Modelling of Parallel Algorithms on Reconfigurable Architectures," Department of Computer Sciences, The University of Texas at Austin, May 1986 (expected).
- John Feo, Ph.D., "Dynamic, Distributed Resource Configuration on SW Banyans," Department of Computer Sciences, The University of Texas at Austin, May 1986 (expected).

6. Verbal Presentations of AFOSR Sponsored Research

- June 26, 1984 - "Software Systems for Parallel Computing, " Conference on Forefronts of Large-Scale Computation Problems, Washington, DC.
- July 15, 1984 - "Software Systems for parallel Architectures," IBM International School on Parallel Processing, Davos, Switzerland.
- August 7, 1984 - "Basic Software for Parallel Processing," Los Alamos National Laboratories - Workshop on Operating Systems and Environments for parallel Processing, Los Alamos, NM.
- October 10, 1984 - "Performance Evaluation of Parallel Architectures," DARPA/Alvey Workshop on Evaluation of High Performance Architectures, London, England.
- October 18, 1984 - "A Fundamental Approach to Structuring of Parallel Computations," Control Data Corporation Symposium on Parallel Processing, Minneapolis, MN.
- January 27, 1985 - "A Unified Approach to Parallel Computation," Yale University, Department of Computer Science, New Haven, CT.

Appendix A

AN EXAMPLE COMPUTATION STRUCTURE AND PROGRAM

This section illustrates the representation concepts described in the previous section. The example is a problem used by J. Dongarra and D. Sorensen of Argonne National Laboratories to test the expressive power of parallel programming languages. The computation is the solution of a triangular matrix

$$Tx = b$$

Dongarra and Sorensen are evaluating the ease with which all of the potential parallelism in the computation can be obtained in the programming system. The method used is to decompose the matrix into blocks as illustrated in Figure 4. The steps in the solution are

a. solve the triangular diagonal blocks

b. execute the transformation

$$T_{ji}x_i - b_j \dashrightarrow b_j$$

on all of the blocks in column i

The schedulable units of computation are:

a. INITARRAY - an initializing routine

b. SOLVE - a block triangular solver

c. MATVECT - multiplies a matrix times a vector and subtracts from another vector in place

The dependency relations are all of type data. The granularity is that of computation data structures. We choose a data driven protocol. Figure 5 gives an explicit computation graph for this computation for NBLOCKS=4 and Figure 6 is the same graph in a proposed indexed notation. Figure 7 is a CSL program where the CONSTRUCT section of the program gives a text string representation of the dependency graph and the executable portion of the program specifies a traversal of the graph.

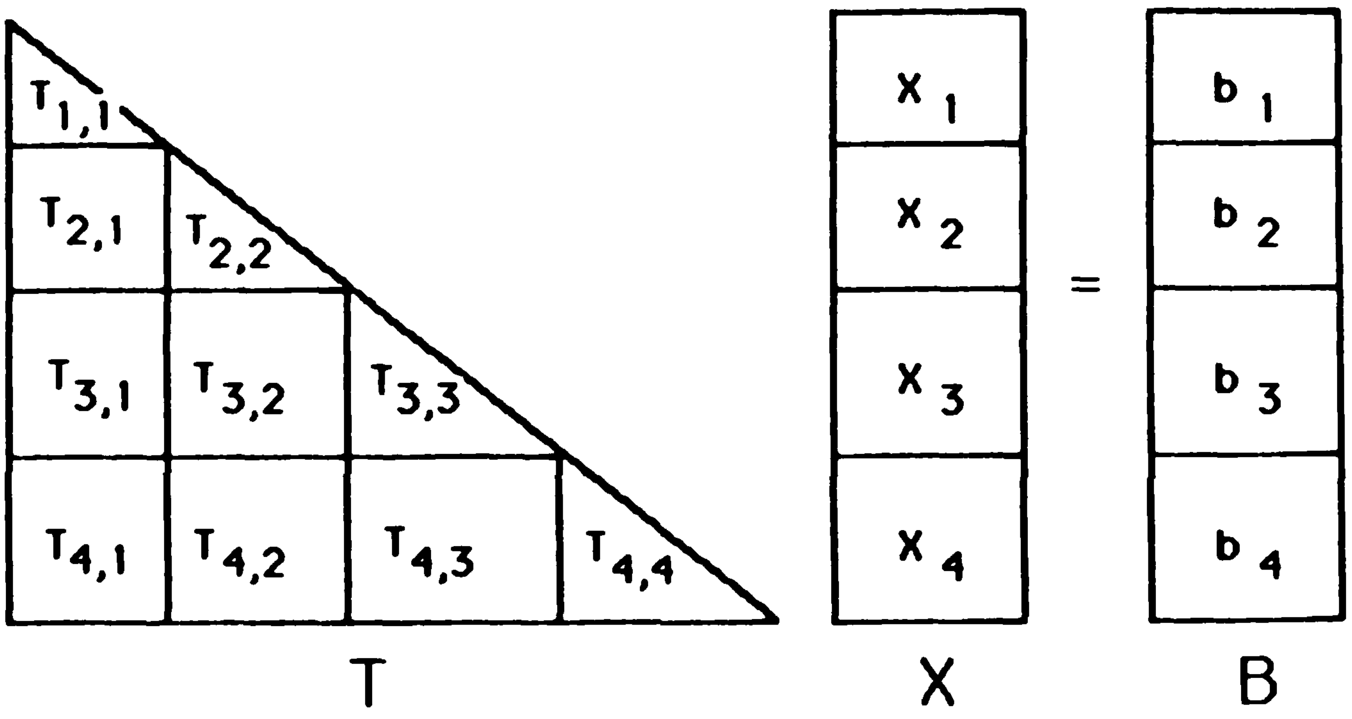
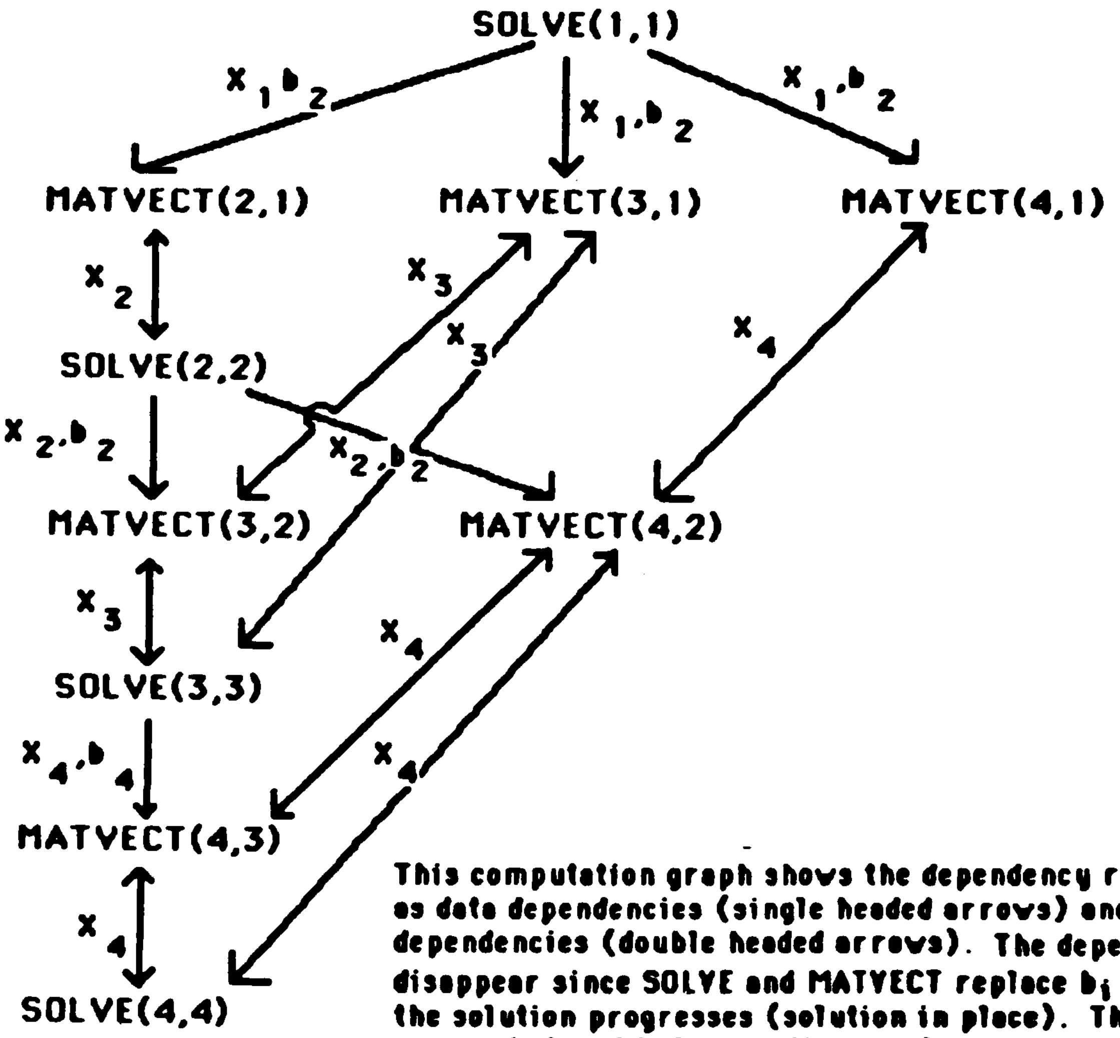


Figure 4 - Block Decomposition of Matrices

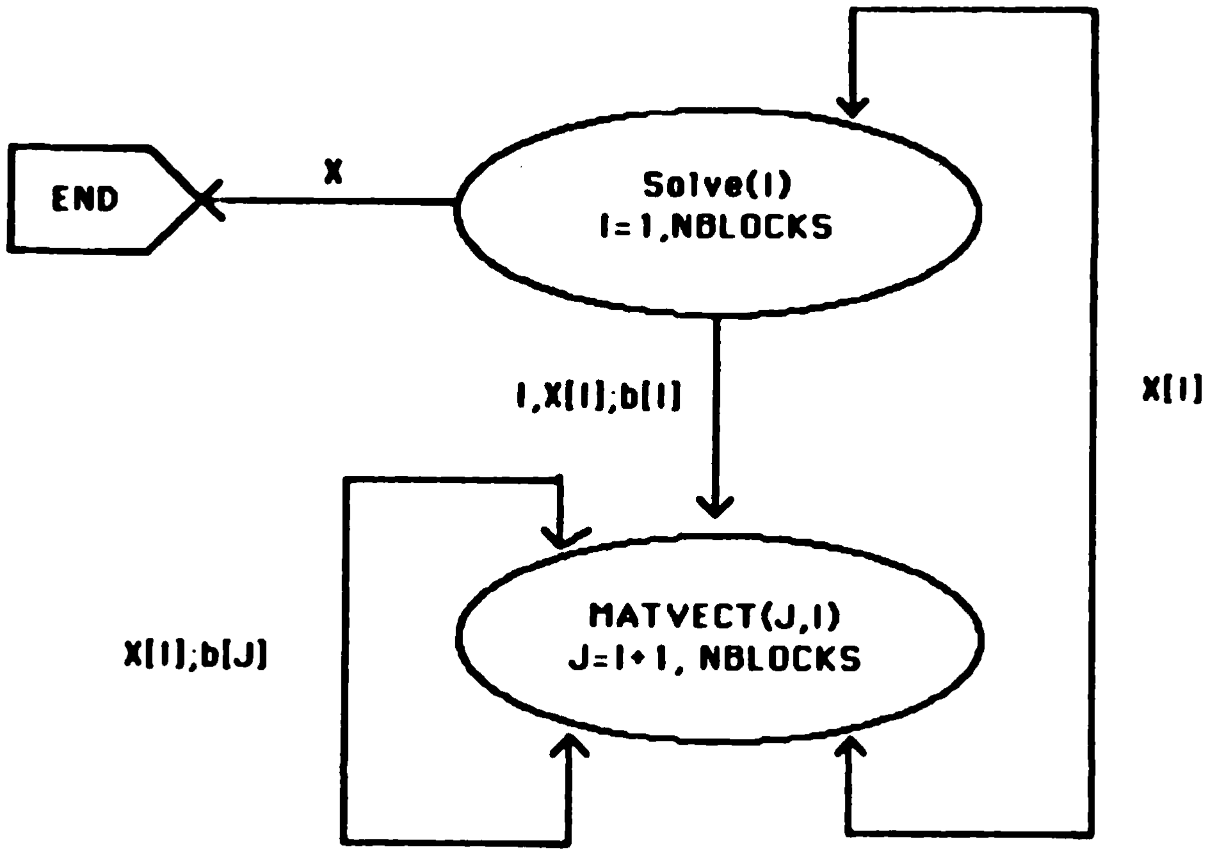


This computation graph shows the dependency relations as data dependencies (single headed arrows) and constraint dependencies (double headed arrows). The dependencies b_i disappear since SOLVE and MATVECT replace b_i by X_i as the solution progresses (solution in place). The $T_{i,j}$ are assumed stored before initiation of execution. Note that all SOLVE's can be simultaneously initiated since they have no input data dependencies. Note that if MATVECT were broken up into

$T_{ji}X_i = y_j$ and $y_j - b_j \rightarrow b_j$

then even more parallelism might be realized but with a more complex graph. SOLVE(1,1) is SOLVE(I) of the CSL program. This is a tradeoff of parallelism for storage.

Figure 5 - A Computation Graph for Parallel Structuring of the Block Triangular Solution



Parallel execution is attempted for all values of indexes. Each instantiation of an indexed SUC occurs when its dependency relations are all satisfied. The mutual exclusion relations are noted by double headed arrows.

Figure 6 - Indexed Form of Computation Graph

```
JOB TRI;

VAR NBLOCKS:  INTEGER;
END;

BEGIN

NBLOCKS := 6;

CONSTRUCT
    TASKS
         INITARRAY : C2 [T(I,J), X(I)] RANGE I = 1 TO NBLOCKS, J = 1 TO I;
                     (* This sets up the values of array T and vector X *)

         SOLVE (I) : C1 [T(I,I),X(I)] RANGE I = 1 TO NBLOCKS;
            (* The code for the diagonal tasks is identical.
               A task is declared for each such block, and they are
               indexed by row.  The compiled code resides in file C1.
               Each task accesses the ith block of vector X,
               and the i,ith block of array T. *)

         MATVECT (I, J) : C3 [ T(I,J), X(I), X(J)] RANGE I = 1 TO NBLOCKS,
                                                       J = 1 TO (I - 1);
            (* The nondiagonal tasks are indexed by row and column.
               Compiled code is in file C2.  Each matvect task
               accesses the ith and jth blocks of vector X, and the
               i,jth block of array T. *)

     CONDITIONS
         TC (I,J) : MATVECT (I,J) RANGE I = 1 TO NBLOCKS,
                                        J = 1 TO (I - 1);
           (* Each matvect task has one task condition associated with
              it, which it can set to communicate with the CSL program.
              The convention used here is that the task conditions are
              initially false, and each task sets them to true as part
              of its execution to signal that it is done. *)
END; (* end construct *)


   WITH T[I,J], X[I] RANGE I = 1 TO NBLOCKS, J = 1 TO I
        DO EXECUTE INITARRAY;     (*initialize X, T *)

   (// BEGIN

        // WAIT TC(I,J) RANGE J = 1 TO I-1;  (* parallel waits. control dc
                                                not advance until all are
```

```
                                                         satisfied. *)
        WITH  X[I] : T[I,I]
            DO EXECUTE SOLVE (I); (* when all matvect tasks finished for
                                     this row, execute diagonal.
                                     The ith block of X is changed, so it
                                     accessed read/write.  The values in
                                     T[i,i] are used in calculations but r
                                     altered, so it is accessed read-only.

        // WITH X[J] : X[I], T[J,I]
              DO EXECUTE MATVECT (J,I)
                                     RANGE J = (I + 1) TO NBLOCKS;
            (* when diagonal task for this column finished, start off
               all matvect tasks in this column in parallel.
               the j,ith task reads X[i] and writes X[j] *)
    END;
     ) RANGE I = 1 TO NBLOCKS;
END.
```

Figure 7 - CSL PROGRAM

This program starts off NBLOCKS parallel streams. Each parallel stream begins by executing a diagonal task first, and then all the matvect tasks in that column in parallel. The start of the actual execution of each stream is coordinated by checking the task conditions of the tasks that must precede the diagonal task execution for that column. The first stream is started immediately, since the range index for the conditions varies from 1 to 0. The second stream executes SOLVE(2) as soon as MATVECT(2,1) is completed, and has communicated that fact to the CSL program. It doesn't have to wait for MATVECT (3,1)..MATVECT (NBLOCKS,1). Since each MATVECT (I,J) puts a lock on the Ith block of X for the duration of its execution, the MATVECTs for each row will have to execute sequentially. This could be remedied by splitting MATVECT into two tasks, the first one reading and computing, and then transmitting the result to the second task, which obtains the necessary write-lock for a shorter period of time.

Note that the range statement which is the last executable statement of the program initiates parallel execution of NBLOCK SOLVE's. The RANGE statement on each MATVECT(J,I) initiates parallel execution of a set of MATVECT tasks. The statement

```
    WITH X[I]: T[I,I]

            DO EXECUTE SOLVE (I))
```

requires SOLVE (I) to obtain exclusive access to X[I] and T[I,I] before executing. Constructing the dependency graph leads naturally to the very simple program of Figure 7. The task condition variables TC(I,J) are used to implement the mutual exclusion relations between the occurrence of SOLVE and the occurrences of MATVECT along a row of blocks. This CSL program selects *one* traversal from many possible traversals. This is one of several places where the current CSL falls short of the generality needed for representation of the computation graphs described in Section 3. Fortran code for the tasks SOLVE and MATVECT as supplied by Dongarra and Sorensen is given in Appendix B. This program has been executed on the experimental version of the CSL programming system on the dual CDC 170/750 configuration in the Computation Center at UT-Austin.

The important element of the graph representation is that it contains no architecturally dependent information. The CSL program again contains no architecturally specific elements. The WAIT and WITH constructs and also the message passing elements which can be used in CONSTRUCT statements can be easily implemented on shared memory or fixed topology architectures.

END
DTIC
8-86