MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo 1249 *July 1990*

The Supercomputer Toolkit and its Applications

Harold Abelson[1], Andrew A. Berlin[1], Jacob Katzenelson[2], William H. McAllister[3], Guillermo J. Rozas[1], and Gerald Jay Sussman[1]

Abstract

The Supercomputer Toolkit is a proposed family of standard hardware and software components from which special-purpose machines can be easily configured. Using the Toolkit, a scientist or an engineer, starting with a suitable computational problem, will be able to readily configure a special purpose multiprocessor that attains supercomputer-class performance on that problem, at a fraction of the cost of a general purpose supercomputer.

The Toolkit is currently being built as a joint project between Hewlett-Packard and MIT. The software and the applications are in various stages of development and research.

**This paper is to appear in the Jerusalem Conference on Information Technology V, October 1990

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of The Department of Defense under the Office of Naval Research contract N00014-89-J-3202.

[1]Artificial Intelligence Laboratory, and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[2]Department of Electrical Engineering, Technion — Israel Institute of Technology.

[3]Hewlett–Packard Corp.

The Supercomputer Toolkit and its Applications

Harold Abelson[1], Andrew A. Berlin[1], Jacob Katzenelson[2], William H. McAllister[3], Guillermo J. Rozas[1], and Gerald Jay Sussman[1]

Abstract

The Supercomputer Toolkit is a proposed family of standard hardware and software components from which special-purpose machines can be easily configured. Using the Toolkit, a scientist or an engineer, starting with a suitable computational problem, will be able to readily configure a special purpose multiprocessor that attains supercomputer-class performance on that problem, at a fraction of the cost of a general purpose supercomputer.

The Toolkit is currently being built as a joint project between Hewlett-Packard and MIT. The software and the applications are in various stages of development and research.

[1]Artificial Intelligence Laboratory, and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[2]Department of Electrical Engineering, Technion — Israel Institute of Technology.

[3]Hewlett–Packard Corp.

The Supercomputer Toolkit and its Applications

1 Introduction

The Supercomputer Toolkit is a proposed family of standard hardware and software components from which special-purpose machines can be easily configured. Using the Toolkit, a scientist or an engineer, starting with a suitable computational problem, will be able to readily configure a special-purpose multiprocessor that attains supercomputer-class performance on that problem, at a fraction of the cost of a general-purpose supercomputer.

Each type of Toolkit hardware module will be implemented as an individual board. The boards fit into a common chassis that furnishes only power and ground. Special cables are used to achieve high-speed communication among boards and to distribute the clock. A user assembles a machine by plugging in the required modules and connecting the cables appropriately. When a particular machine is no longer needed, it can be disassembled, and its modules can be reassembled into other configurations. As of June, 1990, we have designed, fabricated, and are beginning to benchmark the basic Toolkit processor module, tailored for high-performance double-precision floating-point operations. A typical configuration will include several processor modules. Other hardware modules we hope to develop will provide for mass memory and high-speed data-acquisition.

The intent of this arrangement is to make it simple and relatively inexpensive to configure special-purpose computational engines. Yet even if appropriate hardware modules were readily available, these would not be of much use if programming each new machine entailed a major software-development effort, or required an intense analysis to exploit the available parallelism effectively. We believe that, for suitable scientific-computing applications, one can compile extremely high-performance code from high-level languages, and moreover, that the compiler can *automatically* synthesize a pattern of interconnection well-matched to the program being compiled, as well as automatically schedule the computation to make effective use of the available parallelism. In addition to this novel compiler, the software support for the Toolkit will include an assembler, a simulator, and debugging tools. There will also be standard software components, such as a scientific library, for inclusion in Toolkit programs.

We envision that the Toolkit will be used as follows:

One begins with an algorithm that performs the costly inner loop of a computation that is important enough to warrant constructing a special-purpose machine. For example, the simulation part of a multidimensional optimization in the computer-aided design of an analog circuit, or the integration of the differential equations required to achieve the real-time control of a nonlinear process, are appropriate for Toolkit implementation.

The Toolkit software will be used to compile the program, targeted for a number of different Toolkit hardware configurations, some proposed by the user, others generated automatically by the Toolkit compiler itself. The compiler will also produce, for each configuration, a simulation that the user can run on the host machine to help evaluate price-performance tradeoffs. After a configuration has been selected, the user will obtain the required modules, wire them together, and connect the machine he has built to a host computer. The configuration will be verified by means of diagnostics that are automatically generated and loaded from the host. The target program will then be loaded, and the new machine will be ready to be used by host programs as a back-end processor.

2 Historical Motivation

The Digital Orrery [2], constructed in 1983-1984, is a special-purpose numerical engine optimized for high-precision numerical integrations of the equations of motion of small numbers of gravitationally interacting bodies. Using 1980 technology, the device is about 1 cubic foot of electronics, dissipating 150 watts. On the problem it was designed to solve, it was measured to be 60 times faster than a VAX 11/780 with FPA, or 1/3 the speed of a Cray 1S.

The Orrery achieves this performance at modest cost for two reasons. Its communication paths are specialized for the solar-system problem. It is organized as a ring of up to ten processing elements, one for each body to be simulated. The algorithm passes the states of the n bodies around the ring, allowing the computation of all n^2 accelerations in order n time, with negligible communication cost. Additionally, the program that performs the integration completely exploits the data-independence that is inherent in the problem. All available cycles are used for floating-point operations; none are used to support data-structure references.

In 1988, G. Sussman and J. Wisdom used the Orrery to demonstrate that the long-term motion of the planet Pluto, and by implication the dynamics of the Solar System, is chaotic [3]. This required integrating the positions of the outer planets for a simulated time of 845 million years, which required running the Orrery continuously for more than three months. Before the Orrery, high-precision integrations over simulated millions of years were prohibitively expensive, and astrophysicists had done only a few small experiments using carefully scheduled resources.

The objective of our work is to generalize and automate the preparation of such computing instruments. Starting from a mathematical description of an application—for example the equations of motion of the outer planets—a scientist should be able to use the Toolkit to build a modern version of the Digital Orrery in about a week of effort, complete with software. With the same components, and with a similar amount of effort, an engineer should be able to configure a machine, with software, to optimize the design of a high-frequency nonlinear circuit such as a phase-locked loop.

3 Applications

The ability to easily configure special-purpose hardware opens up a variety of important applications that rely upon the ability to perform high-precision simulations in real-time or faster than real-time.

For example, hardware-in-the-loop techniques are used in the development of mechanical systems—the design of a mechanical assembly may be simplified by instrumenting already-designed physical parts and coupling these to actuators driven by simulations of other parts of the assembly. Usually this is done with analog or hybrid computers, but special-purpose digital systems configured from general components could be cheaper, more accurate, and much more flexible.

In the automatic control of highly nonlinear plants, there are techniques that rely upon being able to simulate the dynamics of the plant faster than real time, so as to predict the consequences of proposed control actions. Often it is desirable to operate a plant close to a point of catastrophic failure. The extent to which such control strategies can be safely implemented depends upon the quality of the dynamical model of the plant and upon the speed of computation available to the control engineer. General-purpose computers

with physical characteristics appropriate for use in controllers are inadequate for this use in all but very slow systems.

Alternatively, consider the situation of an electrical engineer optimizing the design of an important nonlinear circuit, such as an analog-to-digital converter. Evaluating each choice of device parameters requires a difficult simulation that needs many hours of time on a workstation-class computer. Typically, the engineer will run a simulation overnight and adjust the parameters after evaluating the result the next day. It is not uncommon for this work to continue for several months. With the Toolkit, the engineer could instead invest a week's effort, analyzing the problem and evaluating alternative Toolkit configurations, to design and configure a special computer to speed up the simulation. If each simulation required half a minute rather than 5 hours, the optimization could be performed using automatic algorithms; in a week of continuous running, a program could achieve a better optimum than manual methods could ever discover.

4 Hardware

The basic Toolkit processor module contains a few arithmetic execution units, a small high-speed multiport memory, and a simple controller. In our prototype, each processor module may connect to other modules via two bidirectional I/O ports, each of which may connect to other units (Each module can connect to about ten others, but we have not yet determined the limits here). All the modules and communication paths of a Toolkit configuration are synchronized by a common clock. One can configure any interprocessor connection graph, within the fan-out limits, by using a processor for every branch, where the interconnections are the nodes (see figure 1).

In our prototype, each board has a peak scalar floating-point speed of 28 double-precision Mflops, and we expect to be able to sustain performance of about half this rate (per board) on real problems. The current design is constructed from off-the-shelf components, and can be easily duplicated at modest cost.

Figure 2 shows the overall structure of the processor module.

Our goal in designing this board was to use the fastest floating-point chips available and to provide enough bandwidth to keep them fully utilized. We chose the two-chip (ALU and multiplier) floating-point chip set made

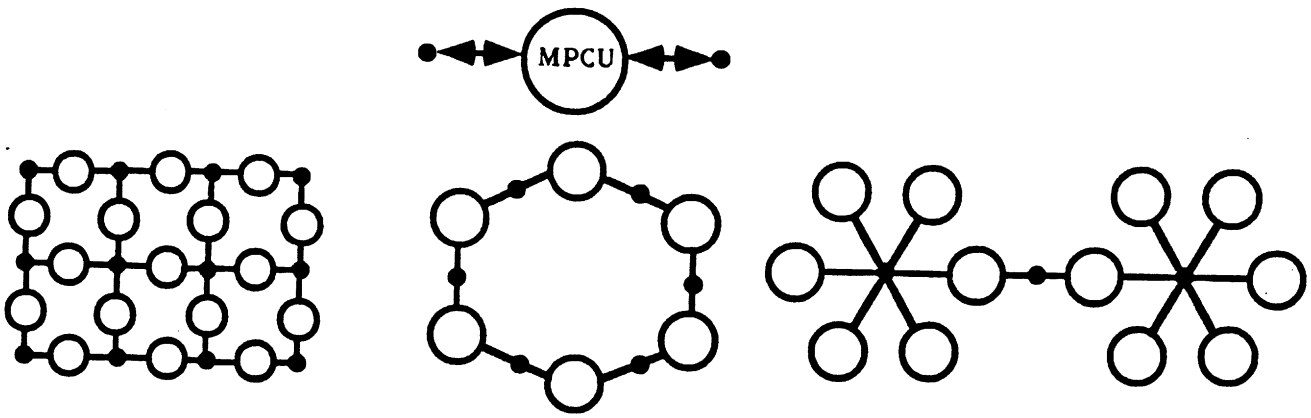

Figure 1: Each processor module has two bidirectional I/O ports. The figure shows how this allows one to build various network architectures: a mesh, a ring and communicating clusters.

by Bipolar Integrated Technologies (B.I.T.) and the fastest easily-available memory (20-ns 16K×4 SRAM).

The floating-point unit (FPU) can multiply two 64-bit arguments during the time it takes to transfer one word from memory. Thus, our desire to obtain a balanced system, in which the FPU is not starved by the memory, required that there should be two separate memories. The memories communicate with the FPU via a 32-entry register array with 5 ports: a read/write port to each memory, two read ports that supply floating-point arguments, and a write port for the floating-point result. The register array is configured from four B.I.T. 5-port 18-bit register-file chips. (This required some clever design and a delicate clocking scheme.) All of the data paths in our prototype are byte-parity protected.

Addresses are supplied to each memory by its own address generator, which was implemented with a 16-bit wide 2901-style microprocessor. Control for the entire processor module is expressed with a very long instruction word—168 bits of horizontal code—that are stored in a 16K deep microprogram memory. The memory is implemented with the same kind of 16K×4

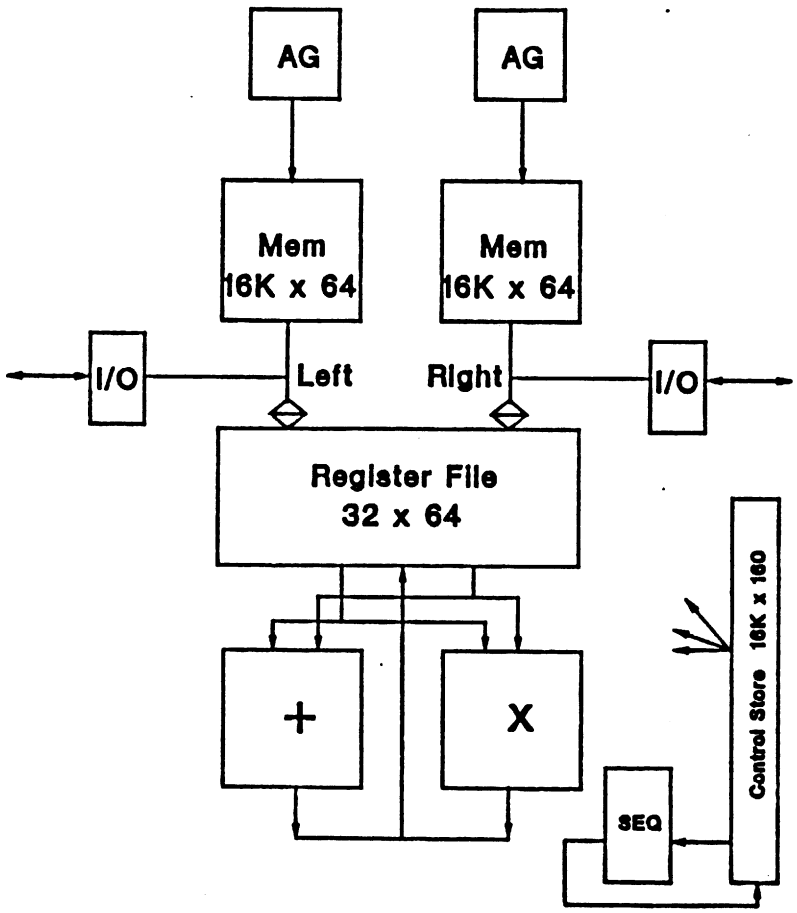

Figure 2: This is the overall architecture of the prototype processor module, consisting of a fast floating-point chip set, a 5-port register file, two memories and address generators, and a sequencer.

SRAMs that we used for the data memories. The microcode memory is addressed using an 16-bit wide 2910-style microprogram sequencer, which also provides limited subroutine and branching capabilities.

We chose the very long instruction word format because during each cycle (about 70 ns) an instruction needs to specify independent operations for the multiplier, the ALU, transfers among the registers, memories, and the I/O ports, an instruction for each of the address generators, and an operation for the sequencer.

Figure 3 shows the layout of the prototype processor module on a 13" $\times$ 15" board, which fits into a standard HP chassis. We expect to assemble a machine with 5 to 10 of these boards during the summer of 1990.

To build a system with several boards we interconnect their I/O ports using controlled-impedance transmission lines, terminated at the ends. Each port can be used to transmit a 64-bit word between processors in two cycles. As there is no hardware arbitration on the I/O ports, it is necessary that the programmer develop a convention for controlling access to each communication channel. To prevent bad programs from burning up the drivers the ports are implemented using open-collector TTL transceivers that can drive impedances as low as 30 Ohms.

To avoid reflections the transmission lines are never branched. They enter the board on one connector, are routed to transceivers and then exit the board on another connector. Careful layout minimizes stubs along the interconnect path. The impedance on the board is the same as the impedance of the ribbon cable used for interconnect.[1]

Since each board has two I/O ports, rearranging cables permits one to statically configure any interconnection scheme (within fanout limits), in which each processor may communicate with two distinct sets of neighbors. For example, figure 4 shows how one uses this scheme to configure a 4-processor cluster.

The entire machine is intended to be a back-end computer that communicates with a host computer via a parallel interface. Communication with the host is significantly slower than communication between boards. Thus, the present prototype is best suited for computations where only a small amount of data is transferred between the Toolkit processors and the host.

[1] Henry Wu was instrumental in developing this interconnect technology for our boards.

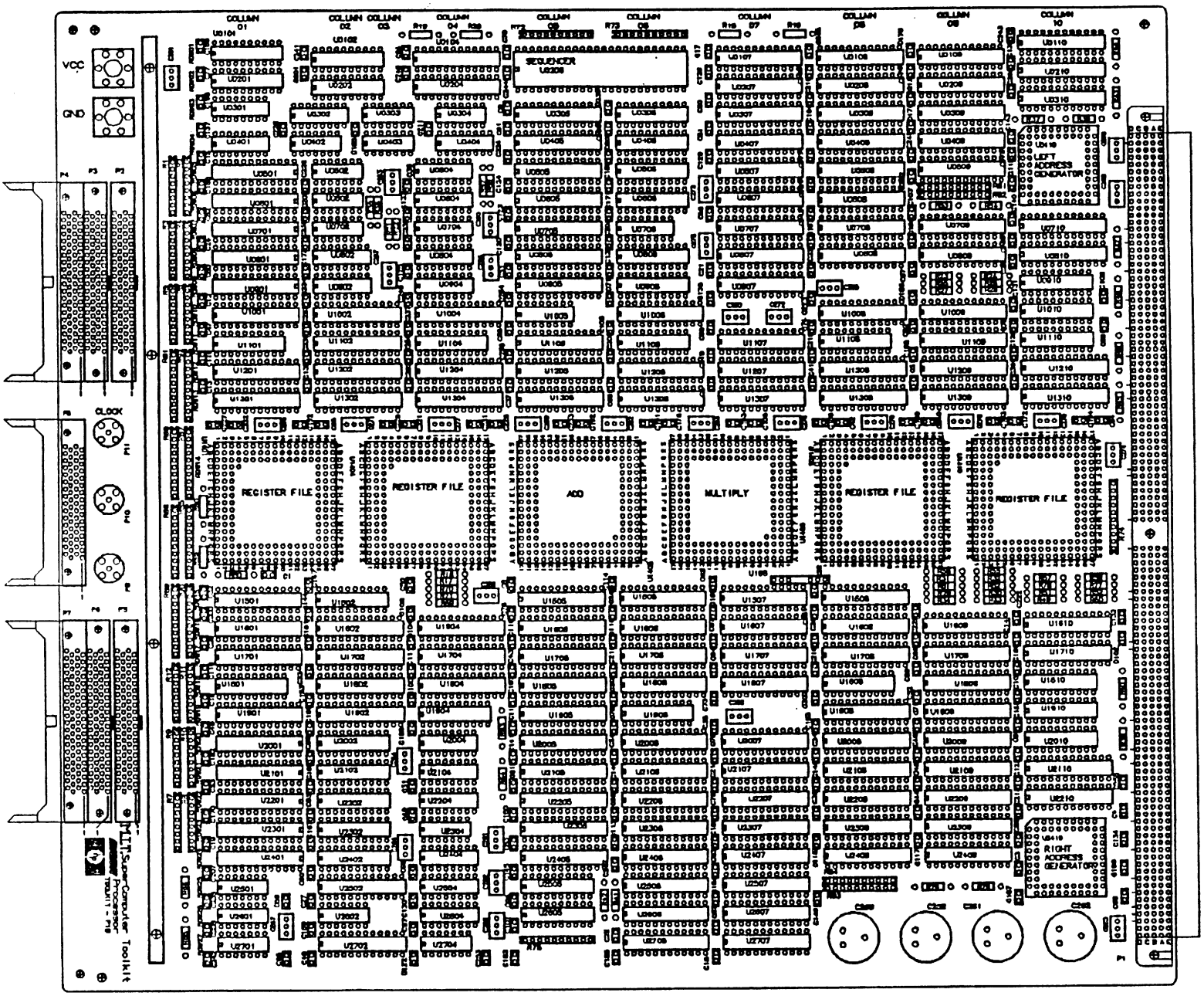

Figure 3: Layout of the prototype on a 13" × 15" board.

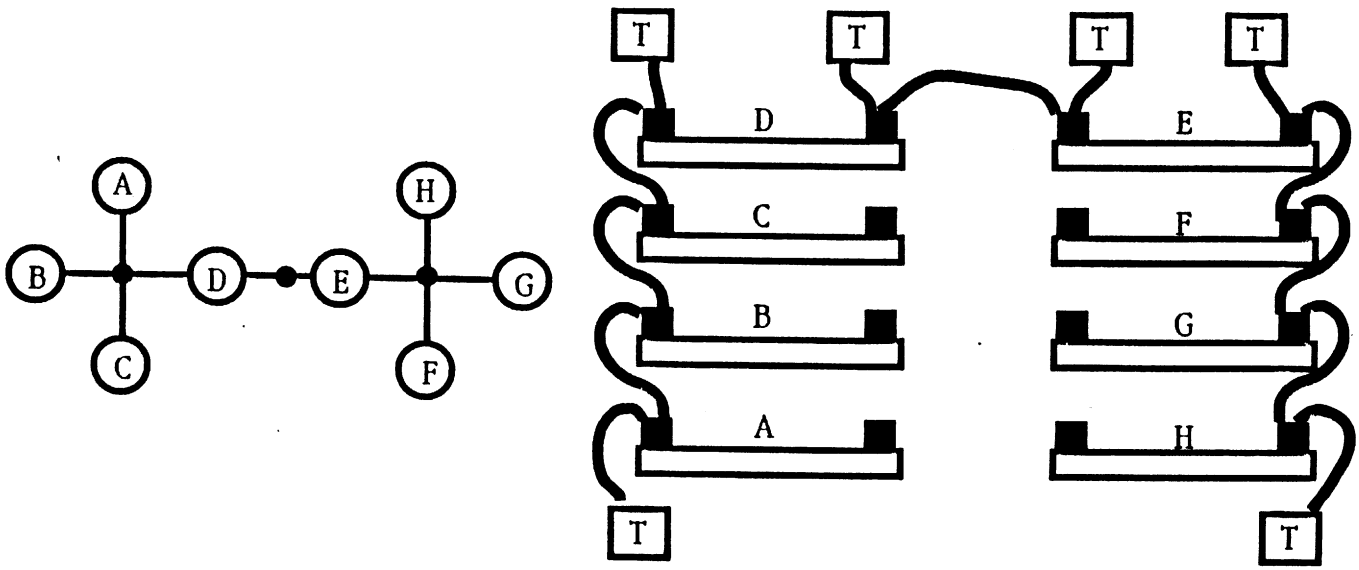

Figure 4: Interconnection between modules is accomplished by transmission lines, allowing one to statically configure any interconnection network in which each processor is connected to at most two nodes. The figure shows how to connect cables to create two communicating 4-processor clusters. The boxes marked "T" are terminators.

5 Software

5.1 Low-Level Programming Model

Each supercomputer Toolkit processor is programmed as a Very Long Instruction Word (VLIW) computer. In every cycle, the following operations can be performed in parallel (see figure 2):

* Two memory transactions, one to left memory and one to right memory. Each memory can perform a load or store operation with the register file on each cycle.

* Two memory address computations, to generate the addresses that will be used to access the memories during the following cycle. The address generators have their own internal register files to support these operations.

* One program-counter operation - conditional branch, jump, call, push/pop, etc.

* One floating-point ALU operation and one floating-point multiply operation. The ALU and multiplier receive their inputs and store their results into the main register-file.

Both arithmetic chips can be operated simultaneously. However, since both the ALU and the multiplier share register-file ports, it is necessary that they do not simultaneously require access to the register-file. For example, while the multiplier is busy doing an operation such as square-root, that takes several cycles to complete, the register-file ports can be used to supply data to the ALU. Operations such as multiply-accumulate use internal feedback paths within the arithmetic chips, thereby freeing up register-file ports.

* Each processor has two I/O ports, each of which is connected to a communication channel. Two cycles are required to transmit a single 64-bit word. Accessing an I/O port uses the internal memory bus for one cycle. Thus, a LEFT I/O operation and a LEFT memory operation can not both be performed during the same cycle.

When multiple processors are to be used for a single application, several programming styles are possible. The simplest style is to have the program counters on all of the boards act in lock-step, effectively forming a multiple board VLIW machine. An alternative is to program the processors in a MIMD style. In the MIMD style, the processors run totally independent programs, exchanging messages via the communication channels as needed.

To support more complex programming styles that combine aspects of

both the VLIW and MIMD styles, the hardware provides a wired-or flag for synchronizing control among multiple boards. For example, in the integration of differential equations, where different state variables have large variations in time scale, it is advantageous to use integrators that admit variable and individual stepsizes. In such systems, some parts of the process can proceed in VLIW fashion, counting out cycles to maintain synchronization, but other parts may need explicit synchronization to keep the individual state variable integrators in step.

5.2 Compilation

We intend to automatically compile and schedule high-performance code for multiple Toolkit modules and automatically generate an appropriate pattern of interconnect, but we have not done that yet. Certainly, the task of programming parallel machines in general is extremely difficult. However, we believe that there are special characteristics of common numerical methods that make automatic scheduling and network generation feasible for a large class of important scientific and engineering applications.

On the other hand, one can make progress using more modest software support. The Orrery was programmed using a fairly simple symbolic microcode assembler. This was possible since the solar-system simulation is not a very complicated program. The partitioning of the problem into processes, the assignment of these processes to processors, and the programming of the connections between processors can be derived from knowledge of the problem.

This kind of low-level programming can be done with the Toolkit now. However, we have developed a compiler that automates the process of building Orrery-like programs. A user specifies, in a high-level language, the straight-line program to be executed in each processor separately. These fragments can be manually glued together to allow simple communication patterns and to construct loops.

The compiler, built by Andy Berlin and Bill Rozas, generates efficient code by using partial evaluation [4, 5] to "flatten" a program. This produces code that contains extremely long straight-line sequences of numerical operations (often several thousand operations long). This makes it feasible to re-order operations to account for pipeline delays, allowing the floating-point units to be fully utilized. In addition, this allows data motion instructions,

such as memory fetches, to be initiated far in advance of the numerical operation that needs the data. Work on the Supercomputer Toolkit compiler has progressed to the point where we can schedule the a solar-system program in such a way as to keep one processor fully utilized. We are now working on generalizing this approach to schedule code for multiple Toolkit processors.

5.3 The Dynamicist's Workbench

Ultimately we expect the Toolkit to be the workhorse for the Dynamicist's Workbench, a tool that will aid scientists and engineers in the simulation and analysis of dynamical systems. The Workbench includes a spectrum of computational tools—including numerical methods and symbolic algebra. These tools are designed so that combined methods, tailored to particular problems, can be constructed on the fly.

For example, one can specify a circuit optimization problem in terms of the circuit diagram. One can investigate the dynamics of a double pendulum in terms of a Lagrangian that describes it. The Dynamicist's Workbench starts with such descriptions and constructs appropriate numerical procedures for simulations and optimizations. It automatically prepares variational equations and sensitivity analysis codes.

Parts of the programs generated by the Dynamicist's Workbench are further compiled by the Toolkit compiler to make microcode for individual Toolkit boards. Other parts of the Workbench code will be used to construct host-interface software and analysis code to be run in the host.

6 Summary

The Toolkit project is not meant to address the difficult issues of large-scale parallel computation. Neither the hardware architecture we propose, nor the interconnection technology, nor in all likelihood our software ideas can be expected to scale to systems with many hundreds of processors. Our goal is to realize means, practical within the limits of current technology, to provide relatively inexpensive supercomputer performance for a limited, but important class of problems in science and engineering. We expect even our prototype implementation to be useful for problems modeled with systems of ordinary differential equations. Additional Toolkit modules that we hope

to develop may make other applications feasible, but we have not discussed applications here that require large memory or for which an appropriate Toolkit configuration would be bigger than a few boards. Simulation of fluid flow is one such example. Other examples can be found in [8].

Efforts with similar goals include the NuMesh effort at MIT, and the iWARP work at CMU [7].

There are other promising strategies for parallel computation, represented by machines such as the MIT Monsoon Dataflow machine, the Connection Machine, the Multiflow computer, and many others. These are general-purpose machines. Our idea differs in that we intend to statically configure both hardware and software for each particular problem. Thus we require no general-purpose software (such as an operating system), no routing protocols, and no hardware to support these features. We believe that when attempting to obtain maximum performance for a fixed level of technology, we cannot afford to pay the price of features intended to support generality.

As a result of its high performance, relative ease of programming and low-cost we expect the Toolkit to have an impact on scientific and engineering computation.

Acknowledgments

The Supercomputer Toolkit is being developed as a joint research effort between MIT and Hewlett-Packard. We are grateful to Joel Birnbaum for his encouragement and support; and to Henry Wu, Robert Grimes, Sam Cox, Darlene Harrell, Karl Hassur, and Dan Zuras for help with the design of prototype processor module. We are especially grateful to John McGrory for his work on the host interface and to Carl Heinzl for his work on diagnostic software.

References

[1] P. Hut and G.J. Sussman, "Advanced Computing for Science," *Scientific American*, vol. 255, no. 10, October 1987.

[2] J. Applegate, M. Douglas, Y. Gürsel, P. Hunter, C. Seitz, G.J. Sussman, "A Digital Orrery," *IEEE Trans. on Computers*, Sept. 1985.

[3] G. J. Sussman and J. Wisdom, "Numerical evidence that the motion of Pluto is chaotic," *Science*, Volume 241, 22 July 1988.

[4] A. Berlin, "Partial evaluation applied to numerical computation", in proceedings of the 1990 ACM Conference on Lisp and Functional Programming. Also see "A Compilation strategy for numerical programs based on partial evaluation," MIT Artificial Intelligence Laboratory Technical Report TR-1144, July, 1989.

[5] A. Berlin and D. Weise, "Compiling Scientific Code using Partial Evaluation," to appear in IEEE Computer. Also see MIT Artificial Intelligence Laboratory Memo number 1145, July, 1989.

[6] H. Abelson and G.J. Sussman, "The Dynamicist's Workbench I: Automatic preparation of numerical experiments," in *Symbolic Computation: Applications to Scientific Computing*, R. Grossman (ed.), *Frontiers in Applied Mathematics*, vol. 5, Society for Industrial and Applied Mathematics, Philadephia, 1989.

[7] S. Borkar, R. Cohen, G. Cox, S. Gleason, T. Gross, H.T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P.S. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iWarp: An Integrated Solution to High-speed Parallel Computing," *Supercomputing '88*, Kissimmee, Florida, Nov., 1988.

[8] B.J. Adler, "Special Purpose Computers," Academic Press, Inc., 1988.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI Memo 1249	2. GOVT ACCESSION NO. AD-A225807	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Supercomputer Toolkit and its Applications		5. TYPE OF REPORT & PERIOD COVERED memorandum
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Harold Abelson, Andrew A. Berlin, Jacob Katzenelson, William H. McAllister, Guillermo J. Rozas, and Gerald Jay Sussman		8. CONTRACT OR GRANT NUMBER(s) N00014-89-J-3202
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE July 1990 13. NUMBER OF PAGES 16
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

special-purpose hardware

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

The Supercomputer Toolkit is a proposed family of standard hardware and software components from which special-purpose machines can be easily configured. Using the Toolkit, a scientist or an engineer, starting with a suitable computational problem, will be able to readily configure a special purpose multiprocessor that attains supercomputer-class performance on that problem, at a fraction of the cost of a general purpose supercomputer.

(con't. on back)

Block 20 continued:

The Toolkit is currently being built as a joint project between Hewlett-Packard and MIT. The software and the applications are in various stages of development and research.

CS-TR Scanning Project
Document Control Form

Date : 10/21/94

Report # AIM-1249

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 16
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☒ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form 2 (PGS) ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

_____
_____
_____
_____

Scanning Agent Signoff:

Date Received: 10/21/94 Date Scanned: 10/24/94 Date Returned: 10/27/94

Scanning Agent Signature: Michael W. Cook