CSAIL

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

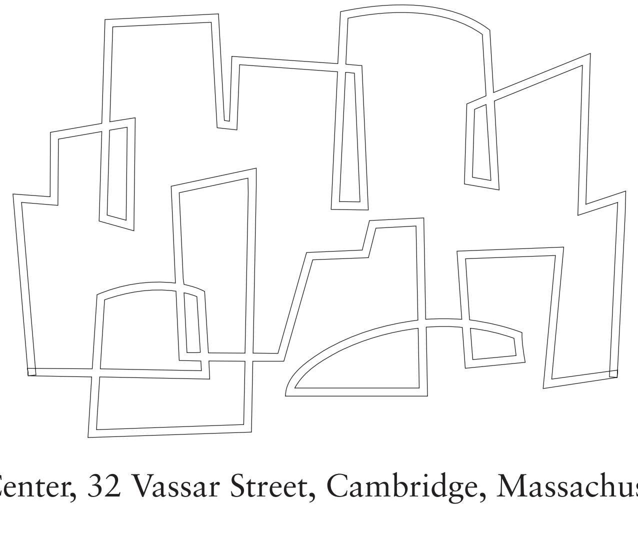
Computation Structures Group Progress Report 1993-94

Y. Zhou, G.A. Boughton

Computation Structures Group Progress Report 1993-94, Y. Zhou (ed.) and G.A. Boughton (ed.)

1994, July

Computation Structures Group
Memo 360



The Stata Center, 32 Vassar Street, Cambridge, Massachusetts 02139

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Computation Structures Group Progress Report 1993-94

Computation Structures Group Memo 360
July 14, 1994

Yuli Zhou (ed.) and G.A. Boughton (ed.)

This report describes research done at the Laboratory for Computer Science of the Massachusetts Institute of Technology. Funding for the Laboratory is provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-92-J-1310.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Computation Structures Group

Progress Report '93–'94

July 14, 1994

Academic Staff

Arvind, Group Leader
J. B. Dennis, Professor Emeritus
G. M. Papadopoulos (on leave)
A. Vezza

Research Staff

G. A. Boughton	R. P. Johnson
C. H. Flood	Y. Zhou

Graduate Students

S. Aditya	D. Chiou	J. Kulik	A. Shaw
S. Asari	K. C. Cho	D. Henry	X. W. Shen
B. S. Ang	S. Coorg	J. W. Maessen	
A. Caro	J. Hoe	G. Rao	
Y. Chery	C. F. Joerg	M. Sharma	

Undergraduate Students

A. Barnard	K. Fynn	W. Lee	D. Panagiotou
W. Chuang	E. Heit	A. Lin	B. Spitznagel
J. Cornez	E. Holley	J. Miao	N. Tender
R. Davis	Y. Koren	E. Ogston	

Technical Staff

J. P. Costanza	R. F. Tiberio

Support Staff

Ann Maderer

Visitors

L. Augustsson	(Chalmers University of Technology, Goteborg, Sweden)
G. Gao	(McGill University, Montreal, Canada)
M. Halbherr	(ETHZ, Zurich, Switzerland)
S. Huet	(Ecole Polytechnique, Palaiseau Cedex, France)
H. Koike	(University of Tokyo, Tokyo, Japan)
J. Morris	(University of Tasmania, Hobart, Tasmania, Australia)
R. Moona	(Indian Institute of Technology, Kanpur, India)
S. K. Nandy	(Indian Institute of Science, Bangalore, India)
J. Stoy	(Oxford University, Oxford, England)

1 Introduction

The Computation Structures Group continues to explore general-purpose parallel computation based on implicitly parallel programming languages, scalable and multithreaded architectures, and compilers and run-time systems for such languages and architectures. Collaboration with industry and applications people has been an important aspect of the group's activities for a number of years.

Monsoon dataflow machines are being used at MIT and other sites [1] for research and teaching in implicit parallel programming though continued maintenance has become increasingly expensive. Due to our shift in focus to the *T project no Monsoon-specific software development was undertaken last year. It has been gratifying to see the programmability and the scalability of the Monsoon architecture. A critical evaluation of Monsoon was published in an article last year [12]. The new Id applications that have been developed on Monsoon are reported in several sections of this report (see Sections 4 and 8).

In the past year we initiated two major transitions in the *T project: one in the area of architectures and the other in the area of languages.

1.1 *T architecture moves from Motorola 88110 to PowerPC

The basic idea behind the *T project is that if a modern RISC processor is augmented with coprocessors to handle the specific needs of parallel processing, such as, communication, remote memory accesses, and frequent synchronization, then it becomes a much better building block for a general purpose parallel machine [35]. After extensive discussions with Motorola, our industrial partner, a concrete implementation of *T based on Motorola's 88110 microprocessor was started in 1992. Since the 88110 had the capability to include additional functional units on-chip, it was decided to build a derivative chip, named the 88110MP. The design of 88110MP included an on-chip network interface unit and hardware support for microthread scheduling [18]. It also maintained upward binary compatibility with 88110. The design of the 88110MP was completed by late 1992 and detailed implementation of the new functional unit was well underway by the July of 1993.

Late in 1993, however, a decision was made to switch from the 88110MP processor to the PowerPC 620 processor. The switch to a PowerPC based *T was motivated by Motorola's commercial shift from the 88K RISC processor family to the PowerPC family. It was anticipated that because of this shift in Motorola's focus it was unlikely that adequate system software or adequate compilers will be available for 88K family of processors. Although Motorola was prepared to finish the 88110MP *T, it was mutually decided that switching to a PowerPC based *T

[1]MIT, Los Alamos National Laboratories, Motorola Cambridge Research Center, University of Southern California, University of Oregon, McGill University

would provide access to much larger software base and access to Motorola's best hardware technology. The switch has also substantially improved the prospect of the commercial availability of the *T machines.

Since it is not possible to make on-chip changes to any PowerPC microprocessor, the switch to a PowerPC based machine has necessitated a large amount of redesign. The 88110MP project taught us that designing derivatives of a state-of-art microprocessor is non-trivial and full of pitfalls. Off-chip additions are a faster, cheaper and a much less riskier way to add functionality. Our new design, called *T-NG, benefited greatly from our experience with the 88110MP *T design. We took the redesign opportunity to include better support for shared global memory than would have been possible with 88110MP [2]. The new architecture and the PowerPC 620 based implementations are described in Sections 5 and 6.

1.2 Id is to be subsumed by pH, a parallel dialect of Haskell

Over the years CSG has invested substantial effort in the design and implementation of the Id programming language. Id is an implicitly parallel language and has a modern non-strict functional language at its core. In the last few years Haskell, designed by a group of international experts including several members from CSG, has emerged as the standard non-strict functional programming language in the research community [34]. In August 1993, a group of people — Arvind (MIT), Lennart Augustsson (Chalmers), James Hicks (Motorola), Simon Peyton Jones (Glasgow), Rishiyur S. Nikhil (DEC), Joe Stoy (Oxford) and John Williams (IBM Research) — met in Cambridge, MA to discuss if it was possible to merge Id and Haskell. pH, a parallel dialect of Haskell, was the outcome of this meeting. pH differs from Haskell in that it follows Id's parallel evaluation strategy, and inherits from Id the I-structure and M-structure extensions.

pH is an attempt to draw together the lazy functional community (as represented by Haskell) and the dataflow community (as represented by Id and Sisal). By sharing a large common language base (Haskell), it is hoped that researchers will find it easier to share ideas and implementations, exchange programs and reduce unnecessary diversity. Application programmers will find it easier to grapple with only one syntax and type system. It should be noted that there are clear differences between pH and traditional Haskell, but having a common framework will allow us to highlight and study these differences more easily than when the languages involved differed more widely.

Detailed implementation plans for pH and its impact on the Id work is discussed in Section 7.

2 Personnel and Visitors

- Arvind received the Charles Baggage Outstanding Scientist Award at the IPPS conference in April, 1994.
- Jan Willem Maessen received the William A. Martin Memorial Prize for M.Eng. thesis in May, 1994.
- Prof. Joe Stoy, returned to Oxford University, England in July, 1993 after visiting CSG for a year.
- Dr. S.K. Nandy of the Indian Institute of Science, Bangalore, India arrived on October, 1993, sponsored by the Indo-U.S. Science and Technology Fellowship Program.
- Prof. John Morris of the University of Tasmania visited from February to June, 1994.
- Prof. Zena Ariola of the University of Oregon visited the group for a week in September, 1993 and for a week in March, 1994.
- Prof. Lennart Augustsson of Chalmers University of Technology, Goteborg, Sweden visited April to June, 1994.
- Prof. Rajat Moona of the Indian Institute of Technology, Kanpur, India arrived in May and will stay through December, 1994.
- Prof. Hanpei Koike of the University of Tokyo is visiting from June, 1994, until March, 1995.
- Prof. Guang-Rong Gao of McGill University in Montreal, Quebec, Canada arrived in mid-June, 1994, and will stay until the end of August.
- Ralph Tiberio joined Exa Corporation in February, 1994.
- Ann Maderer returned after finishing her Masters degree in Radio, Television, and Motion Pictures from the University of North Carolina at Chapel Hill and replaced Lori Avirett-Mackenzie as the administrative assistant.
- Prof. Papadopoulos has been on leave at Thinking Machines Corporation since June 1993.

3 MIT-Motorola Collaboration on Id, pH, Monsoon, and *T

Our collaboration with the Motorola Cambridge Research Center (MCRC) and the Motorola Computer Group in Tempe, Arizona has continued. The results of this collaboration are described throughout this report. Much of MCRC's activities during 1993 were focused on assisting with the definition of the *T architecture and defining aspects of the *T software system. Descriptions of MCRC research are included in the references listed in the Pubications section later in this report.

Some of the important MIT-Motorola collaboration meetings are listed below. Of course this list does not include the frequent informal meetings between CSG and MCRC.

location	**date**
MCRC	July 29, 1993
(phone conference)	August 18, 1993
MCRC	August 30-31, 1993
MCRC	October 18, 1993
Chicago	November 8, 1993
MCRC	December 8, 1993
MCRC	December 14, 1993
MCRC	January 10-11, 1994
MCRC	January 27, 1994
Chicago	March 1, 1994
MCRC	May 18, 1994
MCRC	May 19-20, 1994

4 External Collaborations

4.1 Colorado State University

Wim Bohm's group at Colorado State University studied the producer-consumer parallelism of Eigensolvers composed of the Householder tridiagonalization function, a tridiagonal solver, and a matrix multiplication, and compared the standard top-down Dongarra-Sorensen solver with a new, more space efficient bottom-up version [36]. They verified that the Dongarra-Sorensen solver is more efficient than the more traditional QL algorithm. They showed that in a non-strict functional execution model, the Dongarra-Sorensen algorithm can run completely in parallel with the Householder function. This can be achieved without any change in the code components.

They also studied the NAS parallel benchmark FT [37], which solves a three dimensional partial differential equation using forward and inverse FFTs, written in Id. They compared the performance of three versions of our code written in the various language layers of Id: the functional kernel, the deterministic layer with I-structures, and the non-deterministic layer with M-structures. The I-structure code executes the minimal number of instructions. The M-structure code allows the largest problem sizes to be run at the cost of about 20% increase in instruction count, and 75% to 100% increase in critical path length, compared to the I-structure code.

4.2 Department of Earth, Atmospheric, and Planetary Sciences, MIT

Professor John Marshall's group and CSG are collaborating to develop ocean models on Monsoon and CM5. See Section 8.1 for more details.

4.3 Los Alamos National Lab

Olaf Lubek's group at the Los Alamos National Lab has two versions of MCP-Id, which is a monte-carlo-photon transport code. One is a pure functional version (no I- or M-structures) which has no heap releases and therefore is limited in its problem sizes. The other is a more efficient version, using explicit M-structures, which has releases for all data structures and allows very large numbers of source photons to be run. These codes were developed on the 16-node Monsoon machine at Los Alamos.

They are presently completing an article for submission to the Journal of Functional Programming which describes the pure functional version (as well as a version written in Haskell). The article presents a general description of the problem and its coding, and points to the resource management issues which currently prevent the pure functional version from being used for practical problems.

4.4 McGill University

Researchers at Advanced Compilers, Architectures and Parallel Systems (ACAPS) Lab led by Prof. G. R. Gao of McGill University have used Id system in order to implement a lazy variant of (a subset of) the Id programming language. Their goal was to examine the impact of laziness on fine-grain parallelism, more precisely, to determine whether lazy programs could provide sufficient parallelism to be exploited on fine-grain parallel machines. Their approach used a force and delay style of implementation (using the L-structures of S. Heller), together with an interprocedural form of backwards strictness analysis based on abstract demand propagation. Guy Tremblay is completing his Ph.D thesis in this area and has implemented his abstract demand propagation scheme in the Id compiler.

The researchers associated with ACAPS have also been using the Id language and the Monsoon dataflow machine at McGill to investigate fine-grain parallelism exploitation in non-numerical applications such as the parallel algorithms for VLSI compaction.

4.5 University of Oregon

Prof Zena Ariola and Arvind continue to collaborate on the semantic issues related to Id and pH. They have submitted a paper entitled “Properties of First-order Functional Languages with Sharing” to the journal of Theoretical Computer Science [3]. In the paper a calculus and a model for a first-order functional language with sharing is presented. In most implementations of functional languages, argument subexpressions in a function application are shared to avoid their repeated evaluation. Recursive functions are typically implemented using graphs with cycles. Compilers for these languages sometimes employ non left-linear and left-cyclic rules for optimizations. A Graph Rewriting System (GRS) to address these concerns is developed. It is shown that a GRS without interfering rules is confluent. Along the lines of Lévy's term model for the λ-calculus, a semantics of such a GRS is also presented. An application of the term model to compiler optimizations is discussed.

4.6 University of Tasmania

Dr. John Morris' group at the University of Tasmania are designing a machine based on the *T-NG concepts. This machine will explore some alternatives in the *T architecture and, since it will use existing MPC601 processors, is expected to be available to act as a testbed for *T-NG software. They expect to have an operational system by May 1995.

4.7 Id World

During the last year, R. Paul Johnson distributed Monsoon Id World for Sun SPARCstations to 7 new sites:

- University of Massachusetts Lowell
- Rutgers University, CS Department
- Pusan University
- Naval Research Laboratory
- University Jena
- Technical University in Kosice, Slovakia
- ISIS, Fujitsu Laboratories

This brings the total number of sites which have Monsoon Id World distributions to 38.

5 *T, the Next Generation

Over the past year, we initiated the design of *T-NG (*T, the Next Generation) as a replacement for the 88110MP based *T machine [2]. The machine is being designed and built in collaboration with Motorola. We are close to finishing the high level design of the machine and expect hardware to be available by the end of 1995 or early in 1996.

*T-NG is built around stock PowerPC 620 microprocessors and relies heavily on commercial technology. Unlike the 88110MP-based *T, *T-NG does not modify the 620 microprocessor at all. All the additional support required for parallel processing, notably access to an interconnection network, is provided off-chip. *T-NG also provides some basic support for coherent caching of global (shared) memory. This support is provided with minimum amount of hardware, relying instead on software and the messaging substrate to provide most of the required functionalities.

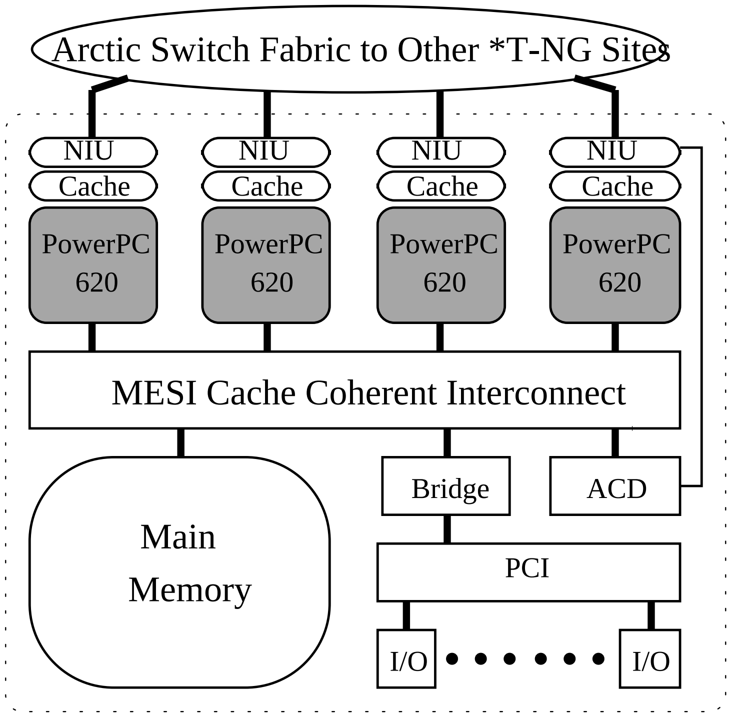


Figure 1: A *T-NG site

*T-NG is organized around *sites* which are connected via a network. Each site consists of several 620 processors, memory and I/O channels connected on a high-bandwidth snoopy bus. A site is therefore capable of functioning on its own as a powerful SMP workstation. Each 620 processor at a site can (but does not have to) have an external *Network Interface Unit* (NIU) that connects the processor to a network built out of Arctic routing chips. We will probably use a fat-tree network in *T-NG. Such a site is shown in Figure 1. The NIU provides support for distinguishing between user and system messages, and protection between different users. The actual number of NIU-equipped 620's at each site can vary. Not every 620 is required to have an NIU. Thus, a site can have as few as only one NIU-equipped 620, which would provide messaging service to all the other processors at the site. A larger number of NIU-equipped 620's will offer higher bandwidth, and lower latency network accesses. For flexibility of experimentation, we expect to populate each site with about four 620 processors, all of which are equipped with NIUs.

To support global cache coherency, one 620 per site is designated as a *Protocol Processor* (pP), whose primary job is to execute cache coherency related tasks. Each site will also contain an *Address Capture Device* (ACD), which is the only hardware support for global cache-coherent memory. The ACD is a simple device that watches addresses on the snoopy bus, responding to operations to global memory by notifying the local pP. The local pP then carries out the necessary actions to complete the bus operation. This will usually require communicating with the pP of the *home* site, the site which owns the accessed global data, via messages. All cache coherency protocol will be implemented in software executing on the pPs. Such an implementation, while paying some performance penalty, allows greater flexibility in experimentation, and can be implemented very cheaply. We expect to experiment with different coherence protocols, and even protocols that are tailored for particular applications. We also expect to experiment with an software L3 cache implemented by the local pP, which can support sophisticated features like automatic or program specified streaming of data and permit sharing of read-only data within a site.

An interesting aspect of shared memory support on *T-NG is its memory model. On *T-NG the virtual memory space on each site is partitioned into local and global spaces. The local virtual address space is private to each site, *i.e.* a local virtual address X refers to different locations when dereferenced on different sites. However, a virtual address Y in the global address space refers to the same location when deferenced on all sites.

This memory model is a hybrid between the conventional message passing view and the conventional shared memory view of memory. Message passing machines provide a separate virtual space on each processor node. Shared memory machines, on the other hand, provide a single virtual space across all the processors that are running a parallel program.

There are two main reasons behind *T-NG's memory model. The first is that most parallel execution models distinguish between local and global address spaces; the local address space does not involve maintenance of cache coherency across sites. Since our global memory is supported mostly in software, this will allow accesses to local space to be much faster than if they were kept coherent across sites. The second is that the design allows virtual memory management, specifically, paging and the maintenance of page tables, to be done independently at each site.

With good support for high-bandwidth, low-latency message passing and flexible support for various kinds of coherent caching of shared memory, *T-NG offers a very wide space for experimenting with different styles of parallel execution. This makes it a flexible machine for further research and experimentation in parallel processing.

6 *T, Implementation

6.1 Hermes

Extensive evaluation of various architectural options for the hardware components which provide the message passing, shared memory and coherent cache capabilities - collectively named 'Hermes' (the messenger of the ancient Greek Gods, not an acronym) was undertaken by a group led by visiting scientist John Morris and consisting of Boon Ang, Derek Chiou, James Hoe, Xiao-Wei Shen and visiting scientist S.K. Nandy. These studies showed that close to optimal performance could be obtained with a configuration containing:

- a processor dedicated to servicing remote memory requests connected directly to one port of an Arctic router,
- an Address Capture Device (ACD) which monitors the site's address bus and captures memory traffic destined for remote sites,
- a Shared Memory Unit (SMU) capable of automatically formatting messages for and receiving messages from remote processors as well as providing significant transaction buffering and
- an Arctic router port dedicated to the SMU.

The performance enhancements obtainable with considerable additional hardware in the SMU could not be justified from an economic standpoint. Thus we opted for an alternative commercially viable configuration (see Figure 1) in which each data processor in a site could have a direct connection to the switch fabric connecting sites, but in which the functions of the SMU will be carried out in software. This configuration will have excellent message passing capabilities as the processors will be connected to the network via their level 2 cache interfaces, providing as close a coupling as is possible using stock commercial hardware. The much simpler hardware will naturally affect the shared memory performance by a small degree, but will allow much more flexibility in experimentation with cache coherence protocols: for instance dynamically adapting protocols are now possible.

6.2 Arctic

Andy Boughton, Jack Costanza, Ralph Tiberio, Tom Durgavich, Doug Faust, and Richard Davis have continued work on the Arctic packet routing chip. Arctic is the primary component in the *T interconnection network. Arctic has four inputs and four outputs. Arctic has high bandwidth (200 MBytes/sec/port), two priority levels, packet sizes up to 96 bytes, and extensive error detection; it has a limited error handling capability, keeps statistics, can directly drive long network links, and provides significant testing support. While Arctic has special features to support fat tree networks, it can also be configured to support any of a wide variety of other staged network topologies.

During the last year a number of changes have been forced on Arctic's design by external forces. The most significant is the change from the original 8x8 configuration to the current 4x4 configuration. This was forced by system packaging changes and tester changes. Originally the *T system was to use multichip modules (MCM's) for packaging the various components. However, MIT's industrial partner on the *T project, the Motorola Computer Group, decided to abandon the use of MCM's for the *T system in favor of more conventional packaging. This decision was motivated by the development cost of MCM's and by a desire to make the *T system more closely correspond to planned Computer Group products. The Computer Group and the chip fabrication vendor, Motorola Commercial Plus Technologies Operation (CPTO), examined alternative packages for Arctic. They concluded that CPTO can not support a package suitable for an 8x8 Arctic within Arctic's design schedule. Motorola CPTO has also changed their testing methodology for ASIC's and their new methodology does not support chips with as many pins as the 8x8 Arctic. For these reasons Arctic's configuration has been changed from 8x8 to 4x4.

Arctic will be fabricated in Motorola H4CP technology rather than Motorola H4C technology. Arctic uses GTL I/O pads and originally Motorola agreed to support these pads in H4C technology. However after the announcement of H4CP, CPTO decided not to allow GTL pads in H4C designs. As a result Arctic's design must be converted from H4C to H4CP. Unfortunately most of the software tools required to finish the design in H4CP were not available before mid May. Unfortunately some of the tools necessary to finish the design in H4CP are not available at the time of this report (first week of June).

A couple of features have also been added to Arctic in response to changes in the *T system design. While an effort has been made to minimize the number of modifications to Arctic, these two features are now important enough to justify their implementation. Arctic's design now guarantees that two packets received on a given input port and destined for a given output port are transmitted in the same order as they were received. This feature may be very useful in supporting global shared memory operations. Another feature has been added that allows an Arctic network to more quickly respond to an operating system crash on an endpoint processor and to more quickly drain packets destined for that processor. This feature should allow the *T system to provide better support for multiple users.

The H4C design for Arctic is fairly complete. A gate level version has been synthesized and partially tested. Preliminary floorplans have been developed for some of the modules including the largest module, the input port.

Currently the design team is finishing the conversion of Arctic to H4CP. Once the conversion is complete, floorplanning of the H4CP design will be done. Testing is in progress. The first pass of place and route should occur in July. Tape out for fabrication of prototype chips should occur by the end of summer.

6.3 Global Cache Coherence

In *T, because cache coherence protocols are incorporated entirely in software, there is considerable flexibility for experimentation and variation in the protocols. Xiao-Wei Shen has been working on designing and evaluating various cache coherence protocols. Different memory consistency models, such as sequential consistency, release consistency and location consistency, are being studied.

The cache coherence mechanism has two components, the home Protocol Processor (pP) engine and the site SMU engine. The MESI protocol of the PowerPC is responsible for cache coherence within a site. For each memory line, there is a directory entry in the home pP, whose state can be either stable or transient, depending on whether or not the current access to this memory line has been completed. All the requests that cannot be serviced are buffered for later processing. For each cache line that has an outstanding request, there is a directory entry in the SMU engine.

Many protocols have been proposed with various buffer organizations, methods for storing directory entries, policies for acknowledging transactions and strategies for forwarding requests. All these scenarios are being analyzed with and without FIFO order in message passing and the presence of an L3 software cache. Strict mathematical proofs are being developed to show that the protocols are free from deadlock, livelock and starvation. An event driven simulator is being built to evaluate the performance of the protocols.

7 Id/pH Compilers

In the past our compiler development efforts have evolved around the Id compiler in Common Lisp. The Id-in-Id compiler was developed during the last two years, intended to replace the Id-in-Lisp compiler, also as a test for using Id in large programs with mostly unstructured parallelism. Due to our transition from Id to pH, we are planning to migrate the Id in Id effort into a pH middle end in either pH or C. Figure 2 illustrates our past and present compiler projects. Our plan for compiler work is essentially to move from the left to the right side of the diagram, and can be summarized as follows:

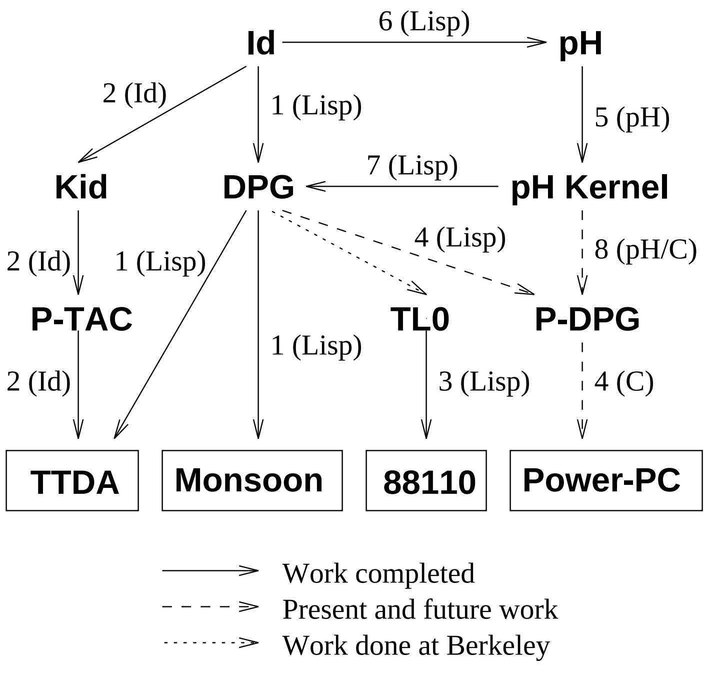


Figure 2: Arrows in this diagram are labeled by the compiler/module and the language in which it is written. The compilers/modules are: 1) The Id compiler in Common Lisp with its TTDA and Monsoon back ends; 2) The Id-in-Id compiler; 3) The 88110 back end; 4) Compiler back end for PowerPC; 5) The pH front end; 6) Id to pH Transliterator; 7) pH kernel to dataflow program graph translator; 8) pH compiler middle end in pH or C.

- The first pH compiler. This consists of the pH front end generating external files in pH kernel format, a reader converting pH kernel files into data flow graph in the Id compiler in Lisp, and the subsequent modules of the Id-in-Lisp compiler. This compiler is nearly completed and can now compile our major benchmarks written in pure Haskell.

- The compiler back end for PowerPC. This includes a partitioning phase in Lisp that generates external files in partitioned-program-graph format, then a module in C that generates native code for IBM PowerPC architecture. Although the code generated is sequential, most of the work is needed for generating code for *T. In addition it will provide us with a fast execution vehicle for pH programs.

- The pH compiler middle end. This part will be written either in pH or in C, connecting with the pH front end and the PowerPC back end via two well-defined external file formats. The middle end itself will contain various optimization modules and a partitioner. It will provide us with a pH compiler that can be integrated on the UNIX platform, freeing us from the traditional dependency on Lisp.

7.1 Id to pH Transliterator

R. Paul Johnson has been working on a tool to translate from Id to pH. This transliterator makes use of the front end of the Id compiler in Lisp and produces a translation in either pH or Haskell syntax. The tool will aid in the transition from Id to pH, and has been used to translate our major benchmarks into pure Haskell. We expect to use the transliterator both to test the pH compiler and to provide the initial translation of the Id-in-Id compiler to pH.

7.2 pH Front End

The front end of the pH compiler takes pH source code, parses and typechecks it, performs some simple optimizations, and translates it into Kernel pH, which can then be used by a pH back end to generate executable code. An initial back end that connects from the pH front end has been written by Yuli Zhou, which will be detailed in the next section. The remainder of the front end has been written in Haskell by Lennart Augustsson. The pH front end is simply a modified version of the front end for an ordinary Haskell compiler which he is writing. Most of the modifications simply add features to the existing compiler. This has a number of potential advantages:

- Compatibility—the compiler will accept valid Haskell programs;

- Interfacing—preexisting support for the module and overloading systems in Haskell and pH;
- Support—The Haskell compiler will be evolving in response to new research in functional programming; for example, it already supports a number of extensions to Haskell's type system. The close integration of the pH and Haskell front ends means that pH can benefit from, and contribute to, much of this research;
- Standardization—The front end will provide a stable base for pH compiler development on all platforms.

Jan-Willem Maessen's Master of Engineering thesis, titled "Eliminating Intermediate Lists in pH using Local Transformations", addresses concerns about Haskell's implementation of arrays. Haskell specifies the contents of arrays by using lists. This is inefficient, and less parallel than the specialized array comprehensions used by Id. A simple set of `map/reduce` transformations exploit the fact that such lists are immediately consumed as they are generated. List traversals are expressed in such a way that a large class of data structures—those expressible using monoids—can be expressed using lists while remaining open to optimization. When used together with code inlining, these techniques should permit the pH compiler to match or exceed the performance of the Id compiler on code involving lists and arrays. In addition, both Haskell and pH programmers stand to benefit from the abstract coding style which the `map/reduce` transformations encourage.

The existing front end has been used to produce working benchmark code. However, there are a few inefficiencies which must be resolved before the front end is put to general use. Lennart Augustsson is working on a code in-liner, which is important in eliminating certain kinds of overloading and reducing the use of higher-order function calls in compiled code. It is also crucial to the standalone `map/reduce` transformation phase written by Jan-Willem Maessen, which is being incorporated into the pH front end. These portions of the compiler will be complete, and naming issues in Kernel pH will be resolved, by the beginning of August. This front end is intended to be used without modification by the future pH compilers developed at CSG.

7.3 Kernel pH to Data Flow Graph

Yuli Zhou has been working on the first pH compiler, which makes use of both the pH front end and the Id compiler in Lisp. Although our plan is to slowly move away from Lisp, this compiler is needed as it will serve as a pH program development environment for the time being, and will eventually enable us bootstrap away from Lisp once the PowerPC back end is available.

This project initially involved writing an output module for the pH front end and defining the pH kernel format, but that part has been taken over by Lennart Augustsson, who is maintaining the output module as an integral part of the front end and is maintaining the kernel document. The pH kernel is a simple language, essentially lambda calculus terms extended with the following constructs:

- Assortment of primitive operators. These operators are being defined as part of the pH kernel format;
- data types (such as list);
- control constructs, namely those for branching, looping and sequencing. Loops and the sequencing construct are specific to pH;
- primitives for I- and M-structure operations, also pH specific.

Array operations are translated in the front end to primitives allocating and operating on a flat vector indexed from 0.

The major part of the work is to connect the pH front end to the Id compiler in Lisp, via translating the pH kernel into the dataflow program graph abstraction in the compiler. In this way the pH front end will be invoked in place of the front end inside the Id compiler, recognizing pH syntax. A preliminary version of the pH kernel to program graph translator, written in Lisp as an Id compiler module, is near completion. With this module the pH compiler is now able to compile and run all our bench marks written in pure Haskell. However, the pH extension to Haskell is still being developed/tested. The compiler is expected to be fully working for GITA and Monsoon in July. In the mean time, some expected efficiency problems due to its Haskell heritage have surfaced, which require tuning the front end, the pH prelude, and the translator itself.

7.4 Data Flow Graph Partitioning

Yonald Cherry and Boon Ang finished the first version of a partitioner that generates partitioned dataflow graph from a dataflow graph representation of a non-strict program. The partitioning is based on Traub et. al.'s Demand and Dependence Set Partitioning algorithm, with simple extensions. The partitioner also includes switches that allow experimentation with language and implementation changes, for example, a language with strict procedure calls, and an implementation where heap accesses do not always have to be split-phase. The partitioner also includes extensive post-partitioning statistics gathering about the static property of the threads. These statistics provide a view of the static characteristics of the threads generated by the partitioner, and the effect of the different switches on these characteristics.

Satyan Coorg has finished his Master's thesis titled "Partitioning Non-strict Languages for Multi-threaded Code Generation". In a *non-strict* language, functions may return values before their arguments are available, and data structures may be defined before all their components are defined. Compiling such languages to conventional hardware is not straightforward; instructions do not have a fixed compile time ordering. Such an ordering is necessary to execute programs efficiently on current microprocessors. *Partitioning* is the process of compiling a non-strict program into *threads* (i.e., a sequence of instructions). This process involves detecting data dependencies at compile time and using these dependencies to "sequentialize" parts of the program.

Previous work on partitioning did not propagate dependence information across recursive procedure boundaries. Using a representation known as *Paths* we are able to represent dependence information of recursive functions. Also, we incorporate them into a known partitioning algorithm. However, this algorithm fails to make use of all the information contained in paths. To improve the performance of the partitioning algorithm, we design a new algorithm directly based on paths. We prove the correctness of the new algorithm with respect to the operational and denotational semantics of the language. Finally, we suggest extensions to the algorithm to handle data-structures and to make use of context information during partitioning.

7.5 Compiler Back End for PowerPC

Boon Ang, Yonald Chery, and Alejandro Caro have been working on the design and implementation of an Id compiler back end for the PowerPC architecture. The initial objective of this project is to combine state of the art *partitioning* technology with a native code generator to yield the fastest possible Id code on a sequential processor. A subsequent goal is to use this compiler as a stepping-stone for porting the Id compiler to the parallel *T-NG architecture.

This project encompasses a large number issues. Several are highlighted in the following sections.

7.5.1 External File Format

A primary design consideration in this project is to make the back end usable by more than one compiler. To this end, we have specified an external file format that completely describes the structure of an Id program emerging from the "middle end" of most compilers. This file format contains information about instructions, partitions, encapsulators, and code blocks, as well as auxiliary data such as types.

In order to make use of the back end, a compiler need only emit this external representation. For example, the Id-in-Id compiler could generate such a file after partitioning a KId graph, and thus, its output could immediately be transformed into executable PowerPC code. The external representation establishes a useful "firewall" between the middle and back ends of the compiler, a division that did not exist formally in previous compilers.

7.5.2 Fast Native Code Generation

There exist versions of the Id compiler targeted to sequential machines (such as the Berkeley TL0 compiler), but usually, these compilers generate C code as their output. This approach is usually adopted for two reasons: it simplifies code generation, and it makes the compiler very portable. However, this approach introduces a degree of inefficiency into the object code. As much as one may think that C is a "high level" assembly language, it is not.

We believe that we can generate native code which is significantly more efficient than C code with very little additional complexity in the back end. We expect to lose some portability, but we are confident that even this can be overcome with proper software engineering practices.

7.5.3 Software Implementation of I-Structures

A key feature of the Id language is its use of synchronizing data structures (I-Structures). On machines like Monsoon, I-Structures were implemented by relying on hardware support. This type of support will not be available on a stock PowerPC based machine, so a pure software scheme must be designed. We are currently investigating several alternatives, evaluating each along a number of design dimensions such as speed, paging behavior, and flexibility.

7.5.4 Compatibility with Unix

In contrast to Monsoon, the Id compiler for the PowerPC will produce code that should run on a standard Unix system. In order to leverage the available technology to the fullest, we have decided to generate normal Unix object files that can be manipulated with standard system utilities. In addition, we have designed the system so that Id programs can call libraries written in C or other standard languages. This has constrained the space of possible representations for Id data structures, but we believe this capability is important enough to warrant the extra effort.

7.6 Id-in-Id Compiler Project

The fate of the Id-in-Id Compiler project has been directly affected by the group's decision to make pH its primary language. The Id programs that make up the compiler have been frozen and no further work on the compiler will be done in Id. These programs now comprise a significant application that the group can now use for testing.

As the Id-in-Id Compiler dies away, Joanna Kulik is planning the development of a new compiler. The compiler is tentatively titled the pH-in-pH compiler, as it is not yet certain how much of this compiler will be written in pH and how much of it will be written in C. The front end of this compiler has already been written in pH and will be shared with the front end to the current pH in Lisp Compiler. The optimizing portion of the compiler, which is not yet written, will use approaches developed for the Id-in-Id Compiler middle-end. These approaches include using graph-rewriting to implement optimizations and using centralized data-structures to synchronize parallel graph-traversals. The pH-in-pH compiler will also include its own dataflow-graph partitioner. The partitioning algorithm it will use will be based on well-known techniques for partitioning. This partitioner will be used to try out new partitioning techniques, particularly those devised by Satyan Coorg in his MS work. After partitioning, the pH-in-pH Compiler will connect with the PowerPC backend, which will also be shared with the pH in Lisp Compiler.

7.7 88110 Back End

Andy Shaw wrote the 88110 back end for the Id compiler during the summer of 1993 to provide an initial implementation of Id for *T. This work built upon the TAM work done at Berkeley. The 88110 back end provided a direct translation for each TAM instruction into 88110 instructions, so the performance of the code emitted was not very high, but the purpose of the implementation was to allow experimentation with different run-time systems and as a simple proof-of-concept.

Code was compiled and run for multi-processor 88110 *T, and the 88110 back end was eventually modified and enhanced for subsequent work with garbage collection for Id.

7.8 Type Reconstructed Garbage Collection

Shail Aditya, Christine H. Flood, James E Hicks (Motorola) completed a study of storage reclamation strategies for *T. They implemented three different memory management strategies on a *T simulator. These strategies included compiler directed storage reclamation (CDSR), type reconstructed garbage collection (TRGC), and conservative garbage collection (CGC). The two GC strategies both used a simple mark and sweep algorithm. They ran four of our ID benchmarks with different memory usage patterns using each of the strategies.

They determined that CDSR used the least number of run time cycles, but it did not reclaim all of the storage. CGC used less cycles than TRGC except when the benchmark used large scalar arrays, then TRGC was more efficient because it didn't need to traverse these arrays. CGC was also capable of missing garbage, although it didn't on any of their benchmarks.

They plan to make TRGC more efficient by including a compiled marking schema to cut the run time cost of type reconstruction and using a compacting GC algorithm to cut the run time cost of storage allocation/deallocation. A compacting collector will allow use of a much simpler memory allocation strategy and will be more efficient then sweeping the entire heap. It may also help performance by improving locality.

8 Id/pH Applications

8.1 GCM

Kyoo-Chan Cho in close collaboration with Professor John Marshall's group in MIT Department of Earth, Atmospheric, and Planetary Science, has been developing a Global Circulation Model (GCM) in Id. GCM is a state-of-the-art ocean circulation model which simulates the currents in an ocean basin. The ocean model to be articulated solves the full three-dimensional Navier-Stokes equations for an incompressible Boussinesq fluid in a highly irregular domain. Development of this model has a multitude of advantages and wide applicability to the simulation of myriad phenomenon in the atmosphere and the ocean.

Two versions of exactly the same algorithm have been implemented: one is written in Id to be run on Monsoon and the other is written in a data parallel Fortran to be run on CM5. Both versions of the code are operational and have been tested to produce the same results. The two groups are jointly performing a critical comparative study of the two implementations in terms of performance and programmability. A paper based on this study is to be submitted to the Journal of Functional Programming by the end of June, 1994.

8.2 Breakout on Monsoon

Sylvain Huet developed a new set of graphic routines for Monsoon and used them to implement a version of the video game Breakout. The purpose of this simple application is to demonstrate the efficiency of the new set of graphic routines, and to give an idea of their speed. The new graphic library is an extension of Alex Caro's work. It provides in Id about fifty Xlib functions of various types: getting standard parameters, creating windows, drawing, using colors, reading events, etc. It is based on the I/O hardware of Monsoon which allows communication to the front-end unix machine during a computation.

Graphic programming in Id is a bit tricky, because of non-strictness: basically, output functions are useless for any evaluation. So the execution of these graphic routines has to be forced by adding barriers. Another disappointment in using graphics in Id is that usually graphic commands have to be executed in a given order: there is no commutativity. A solution is shown in the Breakout program, but it cannot be easily generalized, and works best with monochrome graphics: it consists in using the XOR (exclusive or) function for the drawing, because XOR is commutative. Thanks to this, there are only two barriers in the mainloop of Breakout.

The interactive part of the program calls only the X routine that provides the state of the pointer buttons. It does not show that more interesting routines (for example XNextEvent) are now available.

Tests show that the speed reaches about fifty Xfunctions per second, but that depends on the XServer. However the bandwidth of the Monsoon hardware cannot allow more than one hundred calls per second.

8.3 Spread Sheet on Monsoon

Sylvain Huet is also developing a simplified spread sheet program for Monsoon with a graphic interface. It uses only integers, and each cell can be allocated with an expression using integers, references to other cells and the four basic operations (the program contains a syntax analysis). The interactive allocation uses both mouse and keyboard input, and allows the content of any cell to be edited.

The calculation is very easy because of the properties of dataflow architecture. The main issue in calculating the values for a given number of cells is determining the order of calculation: some cells refer to other cells and thus must be calculated after those cells (the interactive allocation ensures that there are no cycles, so the calculation is always feasible).

By using M-structures to store the value of the cells, the problem disappears: first, the data-structures of the cells to be calculated are emptied. Then the program starts the computation of all these cells, in parallel. Because reading an empty slot suspends until the slot is full, the dataflow architecture finds the proper order of calculation.

The program provides dynamic evaluation: each time an expression attached to a cell is modified, the program determines which cells have been to be recalculated, computes their value, and displays them.

In fact, the graphic interface is not indispensable for this application, but it is very useful to test quickly different kinds of spread-sheets, and it makes the program really convenient.

9 Other Work

9.1 Parallel Computing on Workstations

As commercial microprocessors become increasingly popular in current MPP architectures, high-performance commercial workstations have also received increased attention as cost-effective building blocks for large parallel-processing systems. James Hoe has developed the Fast User-level Network (FUNet) project which is an attempt at constructing an inexpensive workstation-based parallel system capable of supporting efficient execution of message-passing parallel programs. Based on MIT's Arctic network technology, FUNet connects stock-configured commodity workstations with a high-bandwidth packet-switched routing network. The Fast User-level Network Interface (FUNi) is the custom hardware network interface device that provides access to FUNet for both message passing and remote direct-memory-access (DMA) block transfers between parallel peer processes on

FUNet-connected workstations. The FUNi hardware mechanisms allow direct low-overhead user-level network accesses to improve efficiency of parallel execution. Secure and transparent sharing of FUNet among multiple parallel applications is also supported by the FUNi hardware. FUNi will be realized as SBus peripheral cards to allow compatibility with a variety of workstation platforms. The relaxed clock speed (25MHz max.) of SBus allows FUNi to be inexpensively implemented using FPGA parts that are automatically synthesized from hardware description language. SBus's Direct Virtual Memory Access (DVMA) allows FUNi to be implemented as an intelligent and active device to overcome the performance limitations imposed by existing workstation designs.

A detailed design and evaluation of FUNi is covered in James Hoe's Master's thesis. The performance of FUNi and FUNet have been evaluated with the Proteus-based system-level simulation. The encouraging results from the study has led to further efforts in the implementation of the FUNnet hardware during the Summer of 1994.

9.2 Parallel Circuit Simulation

Chris Joerg has been studying parallel circuit simulation. The growing size and complexity of digital systems has increased the need for fast functional simulation to verify the correctness of designs. Parallel simulation is one possible method to meet this need. Of course this will only be successful if there is sufficient parallelism in a circuit. By using hardware and software with low communication costs it should be possible to exploit sufficient parallelism. As a first step, a simple, compiled, gate level simulator has been built to test this hypothesis.

A compiled simulator has been built which is basically a compiler that takes an input circuit and converts it to an executable C program. This compiler accepts circuits which consist of registers, all of which are clocked on a single clock, and simple n-input, 1-output logic gates. This conversion is done in two steps. First the circuit is partitioned into a set of subcircuits, each of which will be simulated by a separate processor. Any signal that passes between subcircuits will need to have its value explicitly sent between nodes. The second step is scheduling. During scheduling the computation that is to be done on each node is ordered and the communication of data between the nodes is arranged. Communication is minimized without overly restricting the available parallelism. Once this is done the circuit description is output as a C program with explicit message passing. This program is then compiled and executed on the CM-5. This simulator has achieved speedups of over 30 on 64 processors.

Since this simple simulator found sufficient parallelism, work has begun on a more elaborate compiled parallel simulator for synchronous designs. This compiler will accept as input a subset of a hardware description language (namely VHDL) ;

this subset will include the constructs accepted by contemporary hardware synthesis tools. The output will be code for a MIMD multiprocessor such as the CM-5. The compiler will treat the input as a fine grain parallel programming language and will operate like current compilers for such languages. It will translate the program into a flow graph, optimize it, and partition it for execution. This simulator will provide designers with significantly more simulation performance than is currently available.

9.3 Threaded-C on CM-5

Michael Halbherr and Yuli Zhou have developed a multi-threaded C package during the past year together with Chris Joerg, who helped with testing/debugging and also developed substantial application code using the package. The package consists of a language (Threaded-C) as an extension to C, and a runtime system (Parallel Continuation Machine or PCM) implemented on top of CM-5's active message layer.

Initially only intended as a compilation target for Id, Threaded-C was subsequently provided as an abstraction for use directly by programmers due to its conceptual simplicity and its flexibility in optimizing scheduling decisions, the latter proved to be critical in developing parallel programs that can scale over its sequential counter-part on CM-5, which has physically distributed memory, typical of most of current scalable parallel architectures. Since there is substantial difference in cost between local and global memory accesses on these machines, the issue of achieving good load balance must be addressed under severe constraint of maximizing communication locality.

Compared to other multi-threaded languages or libraries, the distinguishing feature of Threaded-C is that threads are written in explicit continuation-passing style. Typically, a function would become a few threads in Threaded-C, each stores its arguments in their own *closures*, the counter-parts of the activation frame of the function. This arrangement greatly simplifies the runtime system, as it is not necessary to hack the C stack; More importantly, since a closure can be executed anywhere once it receives all the arguments, it allows ultimate flexibility in optimizing scheduling decisions either by an automatic scheduler or by the programmer.

Current implementation of Threaded-C/PCM embodies a work-stealing scheduler, and is used in porting several real application programs on CM-5 (ray-tracing, protein-folding, monte-carlo simulation of heat transfer and alpha-beta search in Chess) with linear or near-linear speedup over the original sequential code. This work is described in [11] and in Michael Halbherr's Ph.D. thesis [22]. The package also has support for shared data structures. Better language abstraction to hide the explicit continuation-passing will be considered in the future.

9.4 Fast User-Level Interrupts

Dana Henry is investigating fast user-level interrupts for modern microprocessors. Her proposed architecture fully integrates interrupts into the processor pipeline. Synchronous interrupt handlers (for example, division by zero) begin executing in parallel with preceding unfinished instructions, using the mispredicted branch mechanism which already exists in today's microprocessors. Since asynchronous interrupt handlers do not require a view of the computation state consistent with the main code, they can execute in parallel with the main code, exploiting idle instruction issue slots.

The proposed architecture consists of two symmetrical program threads. The envisioned use is for one thread to handle external, asynchronous interrupts, while the other runs the main code. Except for disjoint program counters, the two threads share all registers enabling fast synchronization and communication. The only new processor state consists of two program counters, two sets of user-level interrupt enable bits, and user-level interrupt handler base register. New instructions enable each thread to interrupt itself or the other thread and to deschedule itself, releasing all computational resources to the other thread.

Although most events can be handled by user-level interrupts, a few such as the end of a process' time slot or the arrival of a message destined for a non-resident process, require the attention of the kernel. For this reason, a traditional kernel-level interrupt is also necessary. The kernel views the two threads as a traditional single process which happens to have a two word program counter.

10 Talks

1. Arvind. "Prospects of Ubiquitious Parallel Computing." Hitachi Central Research Lab, Kokubunji, Japan, July 1, 1993. Kyushu University, Kyushu, Japan, July 5, 1993. Osaka University, Osaka, Japan, July 6, 1993. Kyoto University, Kyoto, Japan, July 7, 1993. Chalmers University of Technology, Goteborg, Sweden, October 12, 1993. Distinguished Lecturer, Georgia Tech, November 4, 1993. New York University, November 5, 1993. Distinguished Lecturer, University of Illinois, February 28, 1994. Keynote address at 8th International Parallel Processing Symposium, Cancun, Mexico, April 27, 1994. State of the art talk at ICCSE (International Conference on Computer Systems and Education), Indian Institute of Science, Bangalore, June 24, 1994.

2. Arvind. "Id-in-Id Compiler Project." ARPA/CSTO HPC Software Fall PI Meeting, San Diego, CA, September 27-29, 1993.

3. Papadopoulos. "*T: Integrated Building Blocks for Parallel Computing." Supercomputing '93 Conference, Portland, OR, November 15, 1993.

4. Dennis. "Machines and Models for Parallel Computing." Swedish Institute for Computer Science, Stockholm, Sweden, March 25, 1994.

5. Dennis. "Program Execution Models and Cache Management Strategies." Fourth Workshop on Scalable Shared-Memory Multiprocessors, Chicago, Illinois, April 16, 1994.

6. Dennis. "Program Portability Across Parallel Architectures" Panel. International Parallel Processing Symposium, Cancun, Mexico, April 27, 1994.

7. Aditya. "Storage Reclamation Techniques for Large-Scale Multi-threaded Architectures." ARPA/CSTO HPC Software Fall PI Meeting, San Diego, CA, September 27-29, 1993.

8. Aditya. "Compiler-directed Type Reconstruction and Tagless Garbage Collection for Polymorphic Languages." Indian Institute of Technology, New Delhi, India, January 24, 1994.

9. Aditya. "Tagless Garbage Collection for Id using Run-time Type Reconstruction." NSF/ICOT Workshop on Parallel Logic Programming and its Programming Environments, Eugene, OR, March 4-6, 1994.

10. Aditya. "Garbage Collection for Strongly-Typed Languages using Run-time Type Reconstruction." ACM Conference on Lisp and Functional Programming, Orlando, Florida, June 27-29, 1994.

11. Beckerle, Mike (of MCRC). "The *T Architecture Using PowerPC Microprocessors." ARPA Microsystems/HPC Systems PI Meeting, Seattle, Washington, November 4, 1993.

12. Boughton. "Arctic Routing Chip." ARPA Microsystems/HPC Systems PI Meeting, Seattle, Washington, November 4, 1993. Parallel Computer Routing and Communication Workshop, Seattle, Washington, May 18, 1994.

13. Boughton. "How much Adaptivity in Routing?" Panel. Parallel Computer Routing and Communication Workshop, Seattle, Washington, May 17, 1994.

14. Coorg. "Partitioning Non-Strict Functional Languages." NSF/ICOT Workshop on Parallel Logic Programming and its Programming Environments, Eugene, OR, March 4-6, 1994.

15. Joerg. "Protein Folding on The CM5: work stealing techniques." Scout User's Group meeting, November 15, 1993.

16. Zhou. "Parallel Programming with Continuation-Passing Threads." Boston University Computer Science Colloquium, Boston, MA, May 20, 1994.

11 References

Publications

[1] Shail Aditya, Christine H. Flood, and James E. Hicks. Garbage Collection for Strongly-Typed Languages using Run-time Type Reconstruction. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, June 1994.

[2] Boon Seong Ang, Arvind, and Derek Chiou. StarT the Next Generation: Integrating Global Caches and Dataflow Architecture. CSG-Memo 354, LCS-MIT, February 1994. To appear in the Proceedings of the Dataflow Workshop, Hamilton Island, Australia.

[3] Zena Ariola and Arvind. Properties of a First-order Functional Language with Sharing. CSG-Memo 347-1, LCS-MIT, June 1994.

[4] Arvind, Derek Chiou, and Boon Seong Ang. *T(StarT) the Next Generation: In the Real World. In *Proceedings of the International Conference on Computer System and Education, IISc, 1994*, April 1994.

[5] G. A. Boughton. Arctic Routing Chip. In *Proceedings of the 1994 University of Washington Parallel Computer Routing and Communication Workshop*, May 1994.

[6] Jack B. Dennis. Stream Data Types for Signal Processing. In *Proceedings of the Dataflow Workshop, Hamilton Island, Australia.* IEEE, 1992. (to be published).

[7] Jack B. Dennis. Machines and Models for Parallel Computing. *International Journal of Parallel Programming*, 22(1):47–77, February 1994.

[8] Jack B. Dennis and Guang-Rong Gao. Multithreaded computer architecture: Principles, examples and issues. In R. A. Iannucci and R. H. Halstead, editors, *Multithreading: A Summary of the State of the Art.* Kluwer, 1994. (expected).

[9] George A. Boughton et al. Arctic User's Manual. CSG-Memo 353, LCS-MIT, February 1994.

[10] Debabrata Ghosh and S. K. Nandy. A 400MHZ Wave-Pipelined 8x8-bit Multiplier in CMOS Technology. In *Proceedings of the 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 1993.

[11] Michael Halbherr, Yuli Zhou, and Chris F. Joerg. Parallel Programming Based on Continuation-Passing Threads. CSG-Memo 355, LCS-MIT, March 1994. To appear in *Proceedings of the 1994 Int. Workshop on Massive Parallelism: Hardware, Software and Applications.*

[12] Jamey Hicks, Derek Chiou, Boon Ang, and Arvind. Performance Studies of Id on the Monsoon Dataflow System. *Journal of Parallel and Distributed Computing, special Dataflow Issue*, July 1993.

[13] J. Morris, B. Ang, D. Chiou, J. Hoe, S.K. Nandy, and X. Shen. Hermes: Communication *T:NGS. CSG-Memo 357, LCS-MIT, May 1994.

[14] John Morris. ACD Requirments. CSG-Memo 358, LCS-MIT, June 1994.

[15] S.K Nandy. An Incessantly Coherent Cache System for Shared Memory Multiprocessor Systems. CSG-Memo 356, LCS-MIT, 1994.

[16] V. Natarajan, A. Agarwal, and D. Kranz. Automatic Partitioning of Parallel Loops for Cache-Coherent Multiprocessors. In *Proceedings of the International Conference on Parallel Processing*, August 1993. (V. Natarajan is a researcher at MCRC.).

[17] V. Natarajan and K. Knobe. Automatic Data Allocation to Minimize Communication on SIMD Machines. *The Journal of Supercomputing*, 7:387–415, December 1993. (V. Natarajan is a researcher at MCRC.).

[18] G.M. Papdopoulos, G.A. Boughton, R. Greiner, and M.J. Beckerle. *T: Integrated Building Blocks for Parallel Computing. CSG-Memo 351, LCS-MIT, July 1993. In *Proceedings of Supercomputing '93*, Portland, Oregon, November 15-19, 1993.

[19] Shuichi Sakai, Andrew Shaw, Hiroshi Matuoka, Hideo Hirono, Yuetsu Kodama, Kazuaki Okamoto, Mitsuhisa Sato, and Takashi Yokota. RICA:

Reduced Interprocessor Communication Architecture – Concept and Mechanisms. In *Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing*, pages 122–125, 1993.

[20] Dan Stefanescu and Yuli Zhou. An Equational Framework for the Flow Analysis of Functional Programs. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, June 1994.

Ph.D. Theses

[21] David Chaiken. *LimitLESS Directories: An Integrated Systems Approach to Scalable Cache Coherence.* PhD thesis, Massachusetts Institute of Technology, May 1994. (Professor Papadopoulos was a reader for this thesis from another group.).

[22] Michael Halbherr. *MIMD-Style Parallel Processing Based on Continuation Passing Threads.* PhD thesis, Massachusetts Institute of Technology and ETHZ, May 1994.

[23] Waldemar Horwat. *Active Abstractions.* PhD thesis, Massachusetts Institute of Technology, May 1994. (Professor Arvind was a reader for this thesis from another group.).

[24] Bradley Kuszmaul. *Synchronized MIMD Computing.* PhD thesis, Massachusetts Institute of Technology, May 1994. (Professor Papadopoulos was a reader for this thesis from another group.).

[25] Richardo Telichevesky. *A Numerical Engine for Distributed Sparse Structures.* PhD thesis, Massachusetts Institute of Technology, February 1994. (Professor Papadopoulos was a reader for this thesis from another group.).

[26] Saed Younis. *A High Speed Parallel Lattice Gas Computer.* PhD thesis, Massachusetts Institute of Technology, June 1994. (Professor Papadopoulos was a reader for this thesis from another group.).

Master's Theses

[27] Satoshi Asari. Evaluation of the GTL Technology for Use in the *T Network. Master's thesis, Massachusetts Institute of Technology, May 1994.

[28] Satyan Coorg. Partitioning Non-strict Languages for Multi-threaded Code Generation. Master's thesis, Massachusetts Institute of Technology, May 1994.

[29] J. C. Hoe. Effective Parallel Computation on Workstation Cluster with User-level Communication Network. Master's thesis, Massachusetts Institute of Technology, February 1994.

[30] Jan-Willem Maessen. Eliminating Intermediate Lists in pH using Local Transformations. Master's thesis, Massachusetts Institute of Technology, May 1994.

[31] Gowri A. Rao. Implementation Issues for a Packet Switched Routing Chip. Master's thesis, Massachusetts Institute of Technology, May 1994.

Bachelor's Theses

[32] Eric R. Heit. Design of a Scheduler for the Arctic Nework Routing Chip. Bachelor's thesis, Massachusetts Institute of Technology, Febuary 1994.

[33] Yuval Koren. The Network Monitor: Implementing Fault-Tolerance in the *T Network. AUP/Bachelor's thesis, Massachusetts Institute of Technology, May 1994.

Other References

[34] Paul Hudak, S. Peyton Jones, and Philip Wadler, editors. *Report on the Programming Language Haskell; a Non-strict, Purely Functional Language*, volume Version 1.2. Yale University, March 1992.

[35] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *T: A Multithreaded Massively Parallel Architecture. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

[36] S. Sur and W. Böhm. Analysis of Non-Strict Functional Implementations of the Dongarra-Sorensen Eigensolver. In *Proc ICS'94, Manchester, UK*, 1994.

[37] S. Sur and W. Böhm. Functional, I-Structure, and M-Structure Implementations of NAS Benchmark FT. In *PACT'94, Montreal, Canada*, 1994.