LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

# Computation Structures Group Progress Report 1992-93

Computation Structures Group Memo 359
July 30, 1993

**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

July 1, 1992 — June 30, 1993

## Academic Staff

Arvind *(Group Leader)*
J. B. Dennis *(Professor Emeritus)*
G. M. Papadopoulos
J. Stoy *(Visiting Professor)*
A. Vezza

## Research Staff

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

## Graduate Students

| | | | | |
|---|---|---|---|---|
| S. Aditya | Y. Chery | S. Glim | C. F. Joerg | A. Shaw |
| B. S. Ang | D. Chiou | J. E. Hicks | J. Kulik | |
| S. A. Brobst | K. C. Cho | J. Hoe | D. Kuszmaul | |
| A. Caro | S. Coorg | A. K. Iyengar | M. Sharma | |

## Undergraduate Students

| | | | | |
|---|---|---|---|---|
| S. Asari | R. Davis | T. Klemas | E. Ogston | A. Shah |
| S. Chamberlain | D. Evans | J. Kulik | D. Panagiotou | N. Tender |
| M. Condell | L. Feeney | J. Kwon | G. Rao | M. Tso |
| J. Cornez | R. Gut | J. Maessen | H. Saleeb | K. Yu |
| A. D'Silva | E. Heit | J. Miao | R. Seto | T. Yu |

## Technical Staff

J. P. Costanza    R. F. Tiberio

## Support Staff

L. L. Avirett-Mackenzie

## Visitors and Adjunct Members

| | |
|---|---|
| M. Halbherr | (ETHZ, Switzerland) |
| M. Motomura | (NEC, Japan) |

# 1 Introduction

The Computation Structures Group is interested in general-purpose parallel computation. Our approach incorporates research in:

- a declarative, implicitly parallel language called Id.
- scalable dataflow and multithreaded architectures.
- compilers and run-time systems for Id, targeting dataflow and other architectures.
- applications programs to guide compiler, language, and architecture research.

We reported last year on the initial work on the hardware implementation for *T. This year we have continued this design work. The current structure of a physical *T node includes two 88110MP's, a memory controller, and DRAM. The nodes will be interconnected using a packet switched interconnection network. The detailed design of the 88110MP is being done by our industrial partner, Motorola. The gate level design has been completed and the full custom layout is currently in progress. The memory controller is a standard Motorola chip. MIT is responsible for the design of the network router chip (Arctic) that will be the primary component of the interconnection network. The first draft of the RTL level design for Arctic has been completed. A first draft of the gate level design of each of the modules of Arctic has been synthesized from the RTL design. Motorola is handling the design of the system packaging.

We have distributed Id world for SUN SPARCstations to 31 sites. An Id workshop was held on Nov 15–16, 1992, with attendees from several institutions. An Id council and an Id-user's group was formed.

We have measured the performance of several strategies for frame management on Monsoon. The *T runtime system was designed and a design review document was produced.

A preliminary backend for *T was completed in June. This backend takes output from the Berkeley Id to TL0 compiler, and does a direct translation of TL0 to *T 88110MP instructions. The compiler is tested on all of our major benchmarks.

An interactive debugger for Id programs on Monsoon has been completed. The Id debugger provides two basic sets of tools for debugging Id programs: the first set allows the programmer to control the behavior of executing programs and the second set allows the programmer to inspect the state of executing Id programs. It provides an extensible set of graphic browsers for data-structures, coupled with a novel type-reconstruction algorithm to provide access to the complete state of an Id program.

A dataflow graph partitioning phase was completed in May for the Id compiler in Lisp. The partioning algorithm exploits local and interprocedural dependencies to obtain larger threads, as a necessary step in producing *T code from Id.

We started working on parallel garbage collection techniques for *T. A framework has been developed to compare various GC algorithms.

We completed a study of parallel dynamic storage allocation algorithms for allocating and deallocating heap objects.

The Id compiler in Id development has reached the stage where we can compile four out of five of our major bench-marks. The code produced by the Id compiler in Id is on the same level of optimization as the Id compiler in Lisp. The data-structures and algorithms used in the compiler were specifically designed for parallel execution, using explicit synchronization mechanisms.

We have also made progress in compiling data-parallel programs for dataflow architectures and identifying opportunities for data-parallelism within Id programs.

Finally, we have started research on high performance workstation based parallel-processing systems. We are designing a prototype system in which the workstations are interconnected through user-level network interfaces to a separated low-overhead network. We believe that this approach will support fine grain parallelism more efficiently than existing workstation based systems.

# 2 Personnel and Visitors

Arvind took a sabbatical at the University of Tokyo, where he taught in the Department of Electrical Engineering for the year.

Greg Papadopoulos was promoted to Associate Professor.

In the spring of 1993, Gregory Papadopoulos received the Ruth and Joel Spira Teaching Award.

In June of 1993, Greg Papadopoulos started a two year leave of absence to be chief architect at Thinking Machines Corporation.

Joseph Stoy came to MIT from Oxford Computing Laboratory at Oxford University in England in August 1992 to be a visiting professor for the academic year. He will be visiting with the group until July 1993.

Jamey Hicks completed his Ph.D. in October 1992 and joined the Motorola Cambridge Research Center.

Arun Iyengar completed his Ph.D. in December 1992 and joined HP/Apollo.

Michael Halbherr has been visiting the group since December 1991. He is working on his Ph.D. thesis in conjunction with ETHZ.

Masato Motomura visited the group from August 1991 until November 1992. He is a fellow with the Center for Advanced Engineering Studies.

Noah Rosen received the George C. Newton Prize for best undergraduate laboratory project.

Thomas Klemas was awarded the Charles W. and Jennifer C. Johnson thesis prize.

Neil Tender earned the Department Head's Special Recognition Award for all the work he completed as a Teaching Assistant for 6.004.

# 3 External Collaborations

Our research continues to benefit from the collaborative efforts of our group with researchers from our industrial partner, the Motorola Computer Group, and with researchers from other institutions. In the past year, these efforts have assisted us in the spread of Id to new platforms and in the evaluation and testing of the Id software environment. On November 15-16 we held an Id Workshop preceding Supercomputing '92, which was successful in the promotion and discussion of related research.

Our work with Motorola is discussed throughout this report.

## 3.1 Berkeley

Prof. David Culler at the University of California at Berkeley continues his investigations into multi-threadid machine architectures, and its threadid abstract machine model, TAM. We have been working closely with the Threadid Abstract Machine group to support our Id Compiler in Id project. In the future, we look forward to experimenting with Id on Thinking Machine's CM-5 using Culler's backend.

## 3.2 Colorado State University

Prof. Wim Bohm and Sumit Sur at the Colorado State University have written and carefully analyzed the following algorithms: LU decompostion and matrix inversion, Fast Fourier Transform, Jacobi Eigen-solver, Householder tridiagonalization, QL decomposition, Dongarra-Sorenson Eigen-solver, MCNP simulation, and a variety of sort routines.

These codes have provided the Id/Monsoon community, with a wealth of information on the performance of Id on the Monsoon machine and have helped to keep the two-node Monsoon system at CSU in heavy use.

These codes will be made available via anonymous ftp.

Bohm's preliminary conculsion is that the following are important areas for continued study:

1. Implicit and leak-free deallocation of data structures
2. Copy avoidance
3. Control of parallelism
4. Loop and data distribution

## 3.3 DEC

Dr. Rishiyur Nikhil at the Digital's Cambridge Research Lab is working on a compiler for Id written in Scheme. Initially, the compiler will compile Id to C for subsequent compilation on standard workstations.

### 3.4 Los Alamos

A team at Los Alamos, headed by Olaf Lubek, has completed a code called MCNP-ID, a Monte Carlo photon transport code written in the dataflow language Id. It incorporates complete photon physics, both simple and detailed with user-selectable crossover point. MCNP-ID allows great flexibility in user specified problem geometry, including:

1. geometric cells specified as intersections and unions of planes, spheres, cylinders, and cones
2. cell compositions of single or multiple nucildes of any densities
3. isotropic or unidirectional photon source, with energy distribution specifiable
4. variance reduction techniques, including energy cutoff, cell importances, and Russian roulette

User-specified tally information includes cell collision counts, surface current, and surface flux. These can be binned by energy bands specified by the user.

The code has been tested on various benchmark problems against the Los Alamos Fortran MCNP and all results agree within the MCNP error estimates. Large runs have been made on the 16-node Monsoon at Los Alamos (the biggest run was about 48 hours). The code is available by anonymous ftp to ftp.c3.lanl.gov and going to the directory pub/MCNP-ID. The code exists there, as well as a document describing the code and the benchmark problems.

### 3.5 McGill

Researchers at McGill University are using Id World (Id compiler + GITA) in order to implement a lazy variant of (a subset of) the Id programming language. Their goal is to examine the impact of laziness on fine-grain parallelism, more precisely, to determine whether lazy programs can provide sufficient parallelism to be exploited on fine-grain parallel machines. Their approach uses a *force* and *delay* style of implementation (using the L-structures of S. Heller), together with an inter-procedural form of backwards strictness analysis based on demand propagation and optimization.

## 4 Id General Topics

### 4.1 Id World

During the last year, R. Paul Johnson has distributed Monsoon Id World for Sun SPARCstations to 31 sites. In November, we released Monsoon Id World V2.0 at the Id Workshop. The Id World distribution includes the MINT interpreter, SPLOT statistics viewer, all Monsoon client programs, EMACS editor support and Id World lisp image. The lisp image is built using Lucid's "SPARC Application Environment." Both Monsoon Id World and the application environment are licensed with "shrinkwrap" agreements.

## 4.2 Id Council

Following the Id Workshop, an "Id Council" and id-users (id-users@abp.lcs.mit.edu) mailing list have been established. Both are intended to facilitate community and consensus in the continued evolution of the Id language and systems. Tony Dahbura from MCRC has coordinated the volunteers in the establishment of the council. These are the current members of the Id Council:

- Arvind, MIT (arvind@abp.lcs.mit.edu)
- W. Bohm, Colorado State (bohm@cs.colostate.edu)
- G. Gao, McGill (gao@andy.cs.mcgill.ca)
- J. Hicks, Motorola (jamey@mcrc.mot.com)
- R. Hiromoto, Los Alamos NL (reh@c3serve.c3.lanl.gov)
- J. Kulik, MIT (jokulik@abp.lcs.mit.edu)
- R. Nikhil, DEC (nikhil@crl.dec.com)
- K. Schauser, UC Berkeley (schauser@cs.berkeley.edu)
- J. Stoy, Oxford (stoy@abp.lcs.mit.edu)

# 5 Id Compilers and Runtime Systems

## 5.1 Runtime Systems

During the past year, Derek Chiou finished his masters thesis, wrote two papers and worked on the *T run-time system. The title of the masters thesis is "Activation Frame Management for the Monsoon Processor". It describes the design, implementation and performance measurements of different frame manager strategies used for Monsoon. Load balancing is an integral part of frame management for Monsoon and the simple schemes we tried are also documented.

The title of the papers are "Performance Visualization on Monsoon" and "Performance Studies of Id on the Monsoon Dataflow System." Both papers will appear or have appeared in the Journal of Parallel and Distributed Computing this year. The first describes the hybrid software/hardware approach of statistics collection used in Monsoon. Several examples of the use of the statistics tools are also given. The second paper describes experiments run to determine the performance of Id running on Monsoon. Speedup results as well as comparisons of Monsoon with stock hardware and languages is given.

The *T run-time system was designed along with other graduate students, faculty and staff. Derek implemented the actual memory managers and load balancer. A design review document was produced by the design committee.

## 5.2 *T Backend

Andy Shaw started working on our first version of a *T backend, and finished it in early June. This back-end is meant to be a starting point for a new back-end for *T. It takes output from the Berkeley Id compiler in the form of TL0, and it does a direct translation from TL0 to *T 88110MP instructions. In order to test the compiler, he also implemented a rudimentary run-time system. Presently, the compiler compiles and runs all of the basic benchmarks in our test suite, including matrix multiply, wavefront, merge sort, paraffins, simple, and gamteb. Future work will involve integrating the back-end to test a real run-time system, which will be written and tested by Derek.

## 5.3 Interactive Debugger

Alejandro Caro has completed a master's thesis entitled "A Debugger for Id". The thesis attacked the problem of debugging the implicitly parallel, non-strict Id language on the Monsoon Dataflow processor. The Id debugger provides two basic sets of tools for debugging Id programs: the first set allows the programmer to control the behavior of executing programs and the second set allows the programmer to inspect the state of executing Id programs.

Id programs on Monsoon exhibit a large amount of parallelism, in particular *control parallelism*. This makes their run-time behavior quite difficult to understand at a low level since there are many fine-grained threads of execution active concurrently. Traditional debuggers provide tools such as *single-stepping* and statement-oriented *breakpoints* to control the behavior of sequential programs. These techniques are unmanageable for Id programs, so the Id debugger takes a different approach.

Conceptually, Id programs are modified during compilation so that during execution, they generate a stream of events. Events are generated when "important" things occur in a program, such as procedure invocations, variable bindings, and data structure accesses. Using a *Halt Condition Language* the programmer can specify a set of events which the debugger is to monitor. When the set of events occurs in the event stream generated by the program, the debugger halts the machine. The combination of events and halt condition language provide a flexible mechanism for specifying dataflow-oriented breakpoints.

Once the machine is halted, the programmer is free to inspect the state of the program. A graphical user interface has been incorporated into the debugger, and it provides powerful tools to explore the structure of a program's *call-tree*. The debugger provides an extensible set of graphical browsers for data structures. These browsers are coupled with a novel *type reconstruction* algorithm to provide excellent access to the complete state of an Id program.

## 5.4 Compiler-directed Type Reconstruction

Source Debugging and Garbage Collection for polymorphic languages like Id pose some interesting challenges over conventional languages like C or Pascal. Due to polymorphism, the compile-time types of functions and data objects may not fully describe their run-time type instantiations. This additional type information has to be *reconstructed* at run-time so that a source debugger can display the state of the machine in detail, or a garbage collector can traverse and mark the reachable heap objects correctly. Traditionally, run-time objects carry type-tags to allow such reconstruction which has its space and time overhead.

In Id, run-time objects do not carry any type-tags. Instead, the run-time type of objects is reconstructed using a compiler-directed type reconstruction scheme developed by Shail Aditya and Alejandro Caro. In this scheme, the compiler generates a polymorphic *type-map* for every Id function that records the type of its local and free variables, and the type instantiations of all call-sites within its body. In addition, using a new technique called *type conservation*, the compiler identifies call-sites where type information may be lost or hidden inside polymorphic closures and inserts code to capture and propagate the type information explicitly in the form of encoded *type-hints*. At run-time, type reconstruction is done by traversing the dynamic call-tree and instantiating the corresponding type-maps of the active functions, using the explicit type-hints wherever necessary. This scheme incurs minimal run-time overhead in the form of explicit type-hints, while achieving complete type reconstruction of run-time objects without universal type-tags.

This type reconstruction scheme was implemented in the Id Compiler and is used by the Interactive Source Debugger for Monsoon. Further work in this direction is underway to apply this technique for tagless garbage collection on *T.

## 5.5 Dataflow Graph Partitioning

Generating code from Id source for a multithreaded machine such as *T requires that sequential instruction threads be identified from the intermediate dataflow program graphs produced during the compilation process. Identifying which dataflow graph instructions can be grouped together to ultimately form a thread is called partitioning. For the past year, Yonald Chery has been working on implementing a dataflow graph partitioning phase for the Id Compiler written in Common LISP as his Master's thesis, completed this May.

Since Id is non-strict, the evaluation order of sub-expressions within a procedure is not completely known at compile-time. This is because conflicting scheduling orders for these sub-expression can be obtained depending on the invocation context of the procedure. These static scheduling conflicts, the result of potential dependencies introduced by non-strictness, are what complicate finding lengthy threads for code generated from Id programs.

Partitioning is performed by initially decorating all entry and exit points of a dataflow graph "basic block" with unique names. An entry/exit point can be thought of as an endpoint for an incoming/outgoing potential dependence, that is, a dependence not expressed as a dataflow arc but one that may exist due to dependencies exposed at run-time. Entry point names are called "inlets" and exit point names are referred to as "outlets". These inlet and outlet names are propagated through instructions in the graph. Partitions are created by alternately grouping instructions according to the inlet or outlet names with which they are associated. Partitioning of a dataflow graph ceases when successive passes of inlet and outlet based partitioning find no further opportunities for consolidation instruction partitions.

Further dataflow graph partitioning opportunities can be exposed if information about the functions graph's calling context can be obtained. To this end, an inter-procedural dataflow graph is constructed to link a dataflow graph to its calling contexts. Upon locally partitioning a dataflow graph, inlet and outlet names are recomputed using information provided by adjacent calling and callee graph in the inter-procedural graph. These new names are propagated to other adjacent graphs regions in the inter-procedural graph and expose further local partitioning opportunities.

## 5.6 Loop Optimizations for Monsoon

Boon Ang finished his master's thesis and wrote three papers. The title of his thesis is "Optimization of Loops for Dynamic Dataflow Machines". There are two main parts of the thesis. The first explores ways of implementing sequential loops efficiently on Monsoon, while the second describes automatic strip-mining. Of the three papers that Boon wrote, two were written in collaboration with other members of the group and were mentioned under *T Runtime System section. The third one appeared at FPCA '93 and is based on the first part of his MS thesis, i.e. efficient implementation of sequential loops.

## 5.7 Garbage Collector for *T

Christine Flood began research on parallel garbage collection (gc) techniques as part of the run time system support for ID on *T. She developed a framework which will allow us to experiment with various gc algorithms including: mark and sweep, incremental mark and sweep, conservative mark and sweep, and mark and compact. The framework provides utilities such as processor synchronization, draining the network of ID messages, and determining if a slot in a frame is empty or full.

She developed a program to model an ID heap. This makes it possible to begin implementation and test of various algorithms on the *T emulator even though the ID compiler for *T is not completed yet.

This garbage collection work will build on top of the work already done in explicit storage reclamation, type reconstruction, and resource management.

## 5.8 Heap Managers

Arun Iyengar completed his doctoral thesis on parallel dynamic storage allocation algorithms. These algorithms are used to allocate and deallocate heap objects. The lifetimes and sizes of heap objects are not always known at compile time. The overhead for managing heap objects in languages which utilize dynamic data frequently may be significant. Iyengar's thesis compares parallel versions of several new and previously proposed algorithms. The algorithms were compared using both real implementations and simulations.

He developed three new dynamic storage allocation algorithms which achieve better performance than all previous ones tested. The new algorithms use an approach similar to quick fit for managing small blocks. More complicated methods are used for managing large blocks. *Multiple free list fit I* uses several linked lists for large blocks. *Modified quick fit* uses a concurrent B-tree for large blocks. *Multiple free list fit II* uses several free lists and a concurrent B-tree for large blocks.

Multiple free list fit I is the best algorithm when the set of different large block sizes is big. Multiple free list fit II is the best algorithm when the set of different large block sizes is small. Heap managers using multiple free list fit I, quick fit, and a first fit system with boundary tags have been implemented on Monsoon.

Iyengar experimented with a number of different B-tree algorithms in order to implement modified quick fit and multiple free list fit II. $B^+-$trees store all elements within leaf nodes of a tree. Basic B-trees store all elements within internal nodes of a tree. Maximum concurrency on $B^+-$trees is obtained by using B-link algorithms. However, he found cases where B-link algorithms perform poorly. B-link algorithms cannot easily be adapted to basic B-trees. The best performance on basic B-trees is obtained using an optimistic, top-down restructuring algorithm.

# 6 Id Compiler in Id

In the past year, the Id compiler in Id is mostly the collective work of the "Id-in-Id team": Shail Aditya, Jan-Willem Maessen, Joanna Kulik, Paul Johnson and Yuli Zhou. Professor Joe Stoy (visiting from Oxford) and Proffessor Jack Dennis spend many hours during the weekly Id-in-Id meeting to help clarify the major design decisions made and to oversee the overall progress of the project.

As of now, the compiler compiles four out of five of our major bench-marks: wave-front, matrix multiply, paraffins and simple. The code produced by the Id compiler in Id executes the same amount of instructions in all categories except for identities, the latter mainly due to the fact that the Id compiler in Id has no phase for peep-hole optimization on TTDA graphs. This is not seen as a problem since we are not interested in further optimizing for TTDA, whose performance model is very different from that of *T.

## 6.1 Structure of the Id Compiler

The Id compiler reads Id source files and produces object code for TTDA (The *MIT Tagged-Token Dataflow Architecture*).

### 6.1.1 Internal Representations

During compilation the compiler transforms a program successively through five intermediate representations:

- **AID** (Abstract Id) Close to Id.

- **KID** (Kernel Id) syntactic sugars in AID are removed, complex pattern matching is replaced by simple multi-way branches, and expressions are flattened by introducing temporary variables for each intermediate result.
- **KG** Graph representation of KID, for optimizations.
- **PTAC** (Parallel Three-Address-Code) data structure and closure handling are exposed in terms of primitive operations on the heap.
- **TTDA Graph** machine instructions for TTDA.

### 6.1.2 Functional Modules

- **Parsing** LALR[1] parser in Id. The parser is generated from a modified version of Berkeley YACC, the lexer is produced by FLEX.
- **Desugaring** This is a macro expansion phase in which multi-clause definitions and functions, list and array comprehensions, and other minor "syntactic sugars" are replaced by simpler AID constructs.
- **Scope Analysis** A record called *variable structure* is allocated and partially filled for each distinct identifier, and shared by all occurrences of the identifier. Empty I-structure slots in the record are to be filled by later modules (e.g. type-checking).
- **Translation to KID** The main task of AID to KID translation is the compilation of pattern matching. Unnecessary blocks are flattend and as few temporary variables are introduced as possible.
- **Type-Checking** The type-checker synthesizes a type for every KID syntax tree node, and replaces every overloaded identifier by one of its monotype instances.
- **Translation to KG** Every KID identifier becomes a node in KG, references to the identifier become pointers to the node. KG is a mutable graph structured as nested blocks of nodes.
- **Optimizations on KG** Optimization are performed by mutating the graph, using *traversal overseers* to synchronize the individual rewriting steps. A group of optimizations are implemented: constant folding, common subexpression elimination, fetch elimination, call substitution, etc.
- **Translation to PTAC** KG is translated to PTAC during a traversal of the KID graph Closures and data structures are represented as vector of words in the heap, and their manipulation is expanded into explicit heap allocation, fetch and store operations.
- **Optimizations on PTAC** Similar to KG optimizations. The previous translation phase introduces some redundant nodes, which can be effectively eliminated by optimizations such as fetch-elimination and hoisting.

- **Triggers and Signals** Triggers and signals are needed to complete the control network within a dataflow graph.

- **Translation to TTDA Graph**

  The translation from PTAC to TTDA graph is very much like that from KG to PTAC, except that TTDA graph is built on a simpler abstraction. Every constant definition, function or loop is translated into a code block.

- **Fanout** Certain TTDA instructions for two-phase memory operations can only have one target. This phase simply adds identity instructions (the fanout trees) to such instructions whenever they have more than one target.

- **Assembler** Currently writes the TTDA object code file in standard format.

## 6.2 Implementation

The Id-in-Id compiler is ultimately intended to run on parallel machines, therefore its algorithms and data-structures have been designed for parallel execution. As soon as we leave the realm of purely functional programs, parallel operations must be synchronized. Within the compiler, synchronization is achieved by exploiting fine-grained producer/consumer parallelism, using I-structure and M-structures — it is the only to avoid sequentializing a parallel program.

### 6.2.1 Syntax Trees

Both AID and KID are tree languages. The algebraic type facility of Id is naturally suited for defining such languages. Modules working on syntax-trees are mostly functional, and have ample latent parallelism. Some of the modules can be pipe-lined to reduce the overall critical path.

### 6.2.2 Properties

Most of the properties of the program elements are collected in the variable structure. A structure for storing other properties, such as position index in the source code, is provided on every syntax-tree and graph node. Frequently, the variable structure or the property structure is created with some empty I-structure slots, to be filled later by other modules down the pipeline. I-structures used in this way allow producers and consumers to synchronize without the need of explicit sequential barriers, which usually have adverse effects on the parallelism of the program.

### 6.2.3 Graphs

KG and PTAC share the same underlying graph abstraction. A graph is a collection of nodes. Each node is a storage cell holding a mutable descriptor. Optimizations on graphs are implemented as graph rewriting during graph traversal.

In fact, graph rewriting cannot be implemented by simply mutating the descriptors in the nodes. Instead, it is implemented as a parallel version of the union-find algorithm.

### 6.2.4 Traversal Overseers

Every module that works on graphs needs to perform some kind of graph traversal, sometimes even repeatedly (e.g. during optimizations). A general mechanism is developed to exploit producer/consumer parallelism during graph traversal, which in addition serves as a versatile synchronization tool.

An overseer hides a mutable table of visited nodes. Typically, an empty table is allocated when traversal enters a block within the graph. In the simplest setup, the first visitor will default to be the producer, and later ones only consume the result. Thus a visitor to a node $A$ will first perform a lookup in the table, if an entry is found then it just returns the result in the table; Otherwise it inserts an entry for $A$, computes the result, and fills the entry with the result. The lookup-and-insert operation is atomic, thus each node is involved in the computation only once.

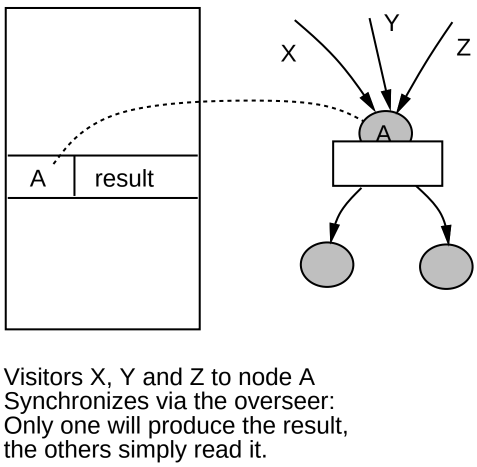


Figure 1: Synchronizing Producer/Consumer with Overseer

In order to accommodate other kinds of visitors, each table entry is implemented as a S-box, which has a mutable status and I-slot for result:

| | | |
|---|---|---|
| node | !status | .result |

The status indicates whether the result slot is full or empty. Upon visiting a node, in addition to perform a lookup-and-insert, different kind of visitors synchronize according to the following protocol:

- Pure consumers immediately read the result slot, regardless of its status;
- Potential producers will always take the status first. If the status is full, it will put back the status and become a consumer by reading the result slot; Otherwise it flips the status to full, and proceeds to compute and fill the result;
- Pure producers always flip the status to full and fill the result (note that in this case we have to make sure that all other visitors are pure consumers).

The traversal overseer provides several primitives to make it convenient for programming in this style.

# 7 *T Hardware

We have continued the detailed design of a hardware implementation for *T this year. For engineering reasons, the structure of a physical *T node has evolved. The current design calls for a single logical *T node to be implemented on a physical *T node. The proposed overall structure of a *T node is shown in Figure 2. As is shown in the figure, the node contains two 88110MP's, a memory controller, and DRAM. The 88110MP's and the memory controller share a 88110 memory bus. One of the 88110MP's is used to implement the Data Processor and Start Processor (of the *T logical node). The other 88110MP is used in conjunction with the memory controller to implement the RMem Processor.

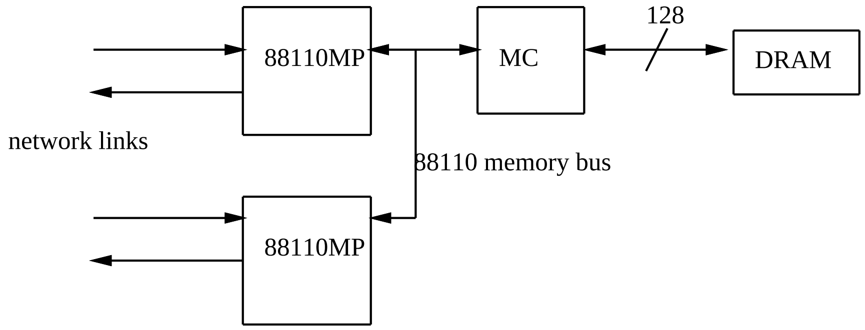


Figure 2: Implementation of a *T node

We have split the detailed design tasks with our industrial partner, Motorola. The design of the 88110MP is being done by Motorola. MIT is responsible for the design of the Arctic network router which forms the basis of the interconnection network for *T.

## 7.1 88110MP

(Much of the 88110MP description below is extracted from the Motorola *T Overview document.)

The 88110MP instruction set is a superset of the 88110 instruction set. The 88110MP includes additional register sets and an additional functional unit which integrate fine-grained communication and synchronization features directly into the instruction set. The new instructions can be dual issued in the same way as other 88110 instructions.

The 88110MP provides a register-set model of network messaging. The user program sees a set of transmit registers into which a message is composed. Issuing a special instruction causes the message to be transmitted from the register set into the network. Message receiving on the 88110MP is also register-set oriented. The arrival of a message from the network can be detected by either an interrupt to the OS or by a poll instruction (which can be executed at user level). A special instruction can be used to move the message into the receive registers. As implied above, the 88110MP allows user mode transmission and reception of messages.

The 88110MP provides hardware support for efficient creation and synchronization of *microthreads*. The *fork* operation creates a new microthread by pushing its source operand microthread descriptor onto the microthread stack. A later *sched* operation will pop this descriptor and begin execution of the microthread by jumping to the IP contained in the descriptor. The *post* instruction behaves like *fork* followed by *sched*. Conditional versions called *cfork* and *cpost* are also provided and they allow testing a semaphore. These instructions can be used to accomplish synchronization.

The 88110MP is being implemented by augmenting the existing full custom design of the 88110. The original 88110 design was done in such a way that it can be modified to include a special additional functional unit. The extra functionality of the 88110MP will be provided by a special functional unit called the Message and Synchronization Unit (MSU). The MSU will connect directly to the primary busses of the 88110.

Motorola has completed the gate level design of the MSU. Much of the full custom layout of the 88110MP has been completed.

## 7.2 Arctic

Andy Boughton, Greg Papadopoulos, Jack Costanza, Ralph Tiberio, roughly 10 undergraduates, and two consultants have worked on the design of Arctic (A Routing Chip That Is Cool). As was discussed last year, Arctic is an eight input eight output packet switch on a chip.

Arctic will support two levels of priority. It will support normal packets and high priority packets.

Most of Arctic will operate off a single 50MHz local clock. However each of the eight input sections will operate off a separate 50MHz clock. Each of these input clocks may have a different phase but all must have a frequency which is extremely close to the local clock.

Arctic will handle packets that range from 128 bits to 768 bits.

All buffering in Arctic is associated with the inputs as is shown in Figure 3. A 768 bit buffer will be allocated for each packet. Obviously in most cases the buffer will not be fully utilized (because the packet will likely be less than 768 bits and because the head of the packet will likely be forwarded before the tail is received).

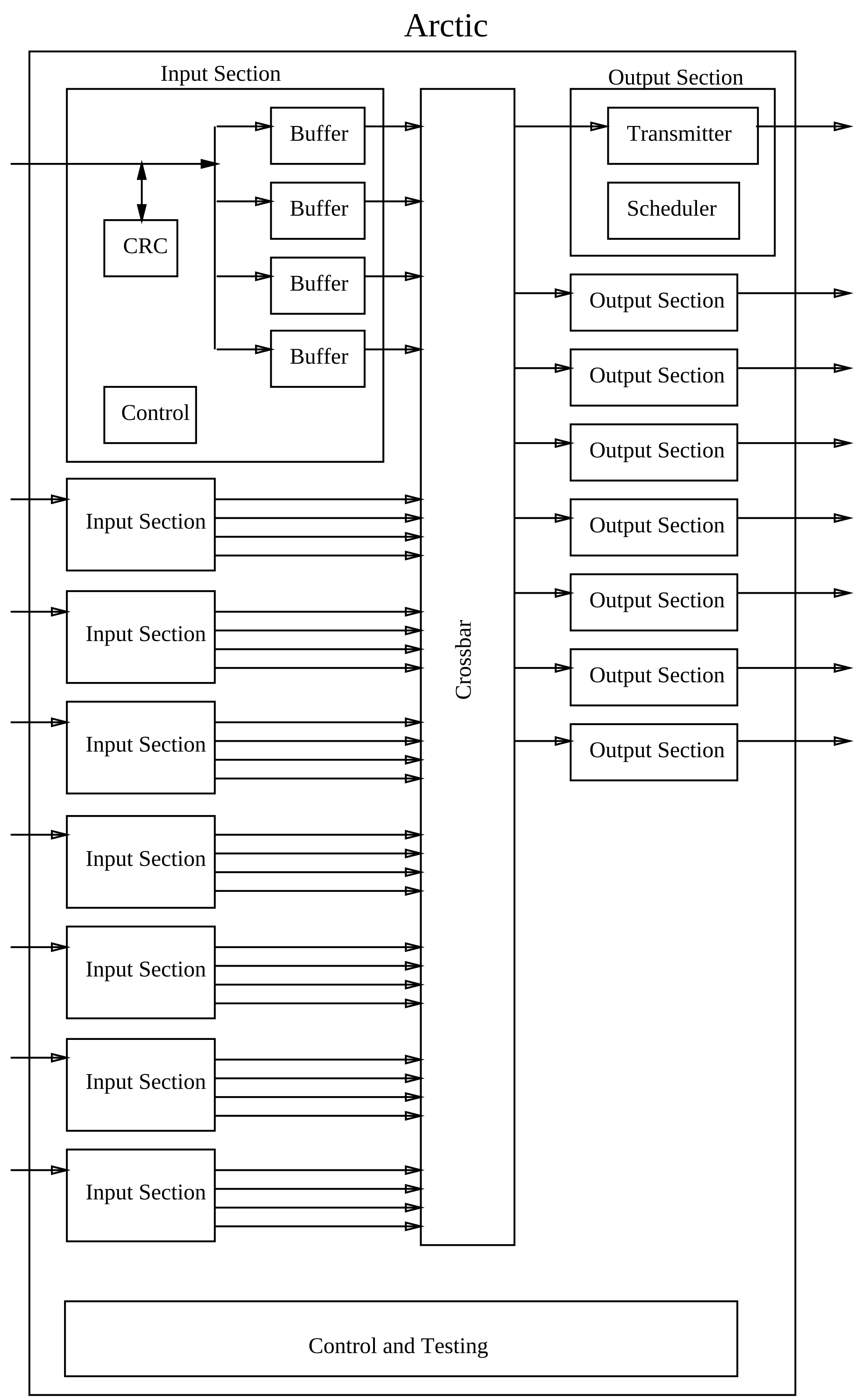


Figure 3: Arctic Block Diagram

The data crossbar in Arctic has an input connected to each buffer of each input section and an output connected to each output section. Thus it can simultaneously transfer packets from any eight buffers to the eight output sections. (For example, the crossbar can simultaneously transfer 3 packets from input section 0 and 1 packet each from input sections 1 through 5.)

Arctic is being implemented on a Motorola gate array. Approximately 200,000 gates will be utilized. The first draft of the RTL level design for Arctic has been completed. The design has simulated correctly for thousands of packets. A first draft of the gate level design of each of the modules of Arctic has been synthesized from the RTL design. We expect to have finished the complete chip design and sent it to Motorola for fabrication before the end of the calendar year.

# 8 Other Work

## 8.1 Compiling Data-Parallel Programs for Dataflow Hardware

Andy Shaw finished his thesis titled "Implementing Data-Parallel Software on Dataflow Hardware", and gave a paper at Frontiers for Parallel Computation titled "Performance of Data-Parallel Primitives on the EM-4 Dataflow Parallel Supercomputer" on October 16.

The primary result of his thesis was that dataflow-style hardware was suitable for running data-parallel software – in particular, the low network-interface overhead of dataflow architectures allows the software implementation of global scan-like operations such as global reduction, one or multi-dimensional scans, broadcast and barrier synchronization. This has implications for the design of computers which are meant to support data-parallel computation in the future: special purpose networks to support global synchronization or scan-like operations are unnecessary in the presence of a high-performance general purpose network with a good processor/network interface.

## 8.2 Data Parallel Computing in Id for *T

Many important scientific computations are amenable to expression in the data parallel computing model and have demonstrated good performance when implemented in this way for massively parallel computers. Professor Jack Dennis together with a few UROP students have begun an effort to support data parallel computing in the Id language for the *T multiprocessor. The work is being done within the framework of the Id in Id compiler. The design of this data parallel capability is based on earlier work he has done for mapping Sisal programs onto the CM-2 Connection Machine.

The data parallel capability will be implemented in three modules to be added to the Id in Id compiler: a Recognizer, a Mapper, and a Generator. The process starts from the Id application program as transformed into its Kernel Id Graph representation by front and middle end modules of the Id in Id compiler.

The Recognizer traverses the Kernel Id tree to find opportunities for data parallel computing. Initailly these will be nested for and while loops that define multidimensional arrays (I-arrays). These nested loops will be transformed into data parallel code blocks, a new form of graph encapsulator to be added to the Kernel Id format.

The Mapper determines the best mapping and alignment strategy for each collection of arrays defined by data parallel code blocks that occur at the same level of loop nesting within the application program.

The Generator module will implement each data parallel code block by constructing code to be run at each node of the multiprocessor partition, including the necessary code for interprocessor communication.

They believe that this data parallel capability will be able to achieve efficient utilization of the *T hardware and provide attractive performance for such codes as the Simple benchmark that satisfy the regularity requirements for data parallel computing.

## 8.3 Parallel Computing on Workstations

After a semester at MIT as a first year graduate student, James Hoe began his master thesis work under the direction of Professor Greg Papadupoulus in the spring semester of 1993. His thesis investigates the possibility of constructing an efficient parallel-processing system based on a cluster of inexpensive commercially available workstations. Existing workstation-based parallel-processing systems are confined to coarse-grained parallelization due to the high overhead cost of interprocessor communication over LAN-type network. To overcome this shortcoming, the thesis proposes a separated low-overhead network dedicated to parallel processing. Each workstation will be connected to the network through a user-level network interface in the form of a peripheral card. We hope to make finer-grained parallelization more efficient on workstation-based parallel-processing systems, and, therefore, attain speed-up similar to those enjoyed by conventional MPP systems.

During the past semester, James began to study various network interface design issues toward reducing communication overhead. To determine the effectiveness of different design options, he has produced a simulator for the proposed system. The Proteus-based software simulator offers sufficient fidelity and performance to execute realistic applications to thoroughly exercise the proposed designs. During this summer, he will continue to work with the simulator to refine the network and network interface design. The final design will be captured in Verilog Hardware Description Language to be synthesized by a hardware compiler. His expects that his master thesis, based on the simulated studies and Verilog-captured hardware design, will come to a conclusion at the end of Fall semester 1993. The actual synthesis of the prototype system will start following the completion of the master thesis.

## Publications

Aditya, S. and Caro, A., "Compiler-Directed Type Reconstruction for Polymorphic Languages," Proceedings of the ACM Conference on Functional Programming Languages and Computer Architectures, Copenhagen, Denmark, June 1993 and Computation Structures Group Memo 348, Laboratory for Computer Science, MIT, Cambridge, MA.

Ariola, Z. and Arvind, "Graph Rewriting Systems: Capturing the Sharing of Computation in Language Implementation," Computation Structures Group Memo 347, Laboratory for Computer Science, MIT, Cambridge, MA, March 1993.

Ariola, Z. and Arvind, "Graph Rewriting Systems for Efficient Compilation," in TERM GRAPH REWRITING: Theory and Practice. Eds. M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eeckelen. Chichester: John Wiley and Sons, 1993. pp. 77-90.

Ang, B.S., "Efficent Implementation of Sequential Loops in Dataflow Computation," Proceedings of the ACM Conference on Functional Programming Languages and Computer Architectures, Copenhagen, Denmark, June 1993 and Computation Structures Group Memo 350, Laboratory for Computer Science, MIT, Cambridge, MA.

Boughton, G.A. and Zhou, Y. (eds.), "Computation Structures Group Progress Report", Computation Structures Group Memo 349, Laboratory for Computer Science, MIT, Cambridge, MA.

Brobst, S.A., Allen, J., and Wilkes, J., "Simulation Model for a Parallel Data Server Architecture," Proceedings of the 1993 Summer Computer Simulation Conference.

Henry, D.S. and Joerg, C.F., "A Tightly Coupled Processor-Network Interface," Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, October 1992 and Computation Structures Group Memo 342, Laboratory for Computer Science, MIT, Cambridge, MA.

Henry, D.S. and Joerg, C.F., "Performance Evaluation of Network Interfaces," Proceedings of the 1992 MIT Student Workshop on VLSI and Parallel Systems, July 1992.

Hicks, J., Chiou, D., Ang, B.S., and Arvind, "Performance Studies of Id on the Monsoon Dataflow System," Journal of Parallel and Distributed Computing, special Dataflow Issue, July 1993 and Computation Structures Group Memo 345, Laboratory for Computer Science, MIT, Cambridge, MA, October 1992.

Motomura, M. and Papadopoulos, G.M., "Local Memory Reference Behavior of Fine-Grain Multithreaded Execution," Computation Structures Group Memo 346, Laboratory for Computer Science, MIT, Cambridge, MA.

Natarajan, V., Chiou, D., and Ang, B.S., "Performance Visualization on Monsoon," Journal of Parallel and Distributed Computing, May 1993.

Nikhil, R.S. and Arvind, "Id: a language with implicit parallelism," in A Comparative Study of Parallel Programming Languages: The Salishan Problems. Ed. John Feo. Amsterdam, New York: North-Holland, 1992. pp. 169-215.

Papadopoulos, G.M. and Molvig, K., "Parallel Architectures for Fluid Flow Simulation," Computing Systems in Engineering, Vol. 3, Nos. 1-4, pp. 261-269, 1992.

Shaw, A., Kodama, Y., Sato, M., Sakai S., and Yamaguchi, Y., "Performance of Data-Parallel Primitives on the EM-4 Dataflow Parallel Supercomputer," Proceedings of Frontiers '92: The 4th Symposium on the Frontiers of Massively Parallel Computation, McLean, VA, October 1992.

## Theses Completed

### S.B.

Chamberlin, S. Design and Imlementation of a Router using a XiLinx FPGA. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Chen, A. Implementation of the Intel 486 SX Microprocessor in Verilog Harware Description Language. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Deo, A. Modeling the Motorola MC88110 Superscalar RISC Microprocessor in Verilog HDL. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Klemas, T. Monte Carlo Simulation of Radiation Transport for Benchmarking Intel's PSC2, iPSC860, and Touchstone Delta Machine. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Kwon, J. Design of a Ticket-Based Scheduler for a Network Router Chip. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Quintana, F. A Digital Answering Machine Using Analog Caller ID. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Rao, G. Transmission and Timing Issues for Arctic Transmit and Receive Apparatus. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Rosen, N. Design of High Speed Interconnects for a "Continuous Computing" Environment. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Saleeb, H. A Memory Network Controller For A Parallel RISC Multiprocessor. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Tender, N. A Microcoded Implementation of the ALPHA(NAUGHT) ISA for the MAYBE Computer. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Ying, C.F. Parallel Lexial Search in Speech Recognition. Bachelor's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

### S.M.

Ang, B.S. Optimization of Loops for Dynamic Dataflow Machines. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Caro, A. A Debugger for Id. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Chery, Y. Dataflow Graph Partitioning for the Id Compiler. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Chiou, D. Activation Frame Memory Management for the Monsoon Processor. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Cho, N. A Simple Fuzzy Logic Buffer Controller and its Digital Hardware Implementation. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

Shaw, A. Implementing Data-Parallel Software on Dataflow Hardware. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1993.

**Ph.D.**

Hicks, J.E. Compiler-Directed Storage Reclamation Using Object Lifetime Analysis. Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, October 1992.

Iyengar, A.K. Dynamic Storage Allocation on a Multiprocessor. Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, December 1992.

## Theses in Progress

Brobst, Stephen A. "Storage Management in a Multi-Threaded Computer Architecture," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected February 1994.

Hoe, James "Parallel Computation on Workstation Cluster Interconnected with a User-Level Network," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected December 1993.

Norton, Joe. "The Design and Evolution of a Hybrid Performance Tool: PROFITEER," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected September 1993.

Shail Aditya "Functional Interfaces for Imperative Programs," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected June 1994.

Sharma, Madhu "Design of a Multithreaded Processor Architecture," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected December 1993.

# Lectures

Arvind. "Prospects of Ubiquitious Parallel Computing." Japan Electronic Industry Development Association, Tokyo, Japan, Jan 28, 1993; Symposium for 20th Anniversary of the IIAS, Fujitsu Labs, Makuhari, Chiba City, Japan (Keynote speaker), Feb 9, 1993; University of Oregon, Eugene, OR, February 22, 1993; MIT-ILP, Tokyo, Japan, May 18, 1993; RWC Labs, Tsukuba, Japan, May 28, 1993; Sony CSL, Tokyo, Japan, June 25, 1993; Hitachi Central Research Lab, Kokubunji, Japan, July 1, 1993.

Arvind. "Why most machines in use today are not parallel machines." Fujitsu Labs, Kawasaki, Japan, August 28, 1992; University of Tokyo, Hongo Campus, Tokyo, Japan, October 13, 1992; Parallel Processing Society of Korea, RIACT, Seoul National University, Seoul, South Korea, October 26, 1992; Sung Kyun Kwan University, Suwan, South Korea, October 27, 1992; Pusan National University, Pusan, South Korea, October 30, 1992.

Arvind. "Why Dataflow and Multithreaded Architectures." University of Tokyo, Hongo Campus, Tokyo, Japan, October 20, 1992; Parallel Processing Society of Korea, RIACT, Seoul National University, Seoul, South Korea, October 26, 1992; Fujitsu Labs, Kawasaki, Japan, November 5, 1992.

Arvind. "Functional Programming in Id." University of Tokyo, Hongo Campus, Tokyo, Japan, November 10, 1992; Fujitsu Labs, Kawasaki, Japan, December 18, 1992.

Arvind. "M-Structures: Programming with State and Non Determinism." University of Tokyo, Hongo Campus, Tokyo, Japan, November 24, 1992.

Arvind. "Static and Dynamic Dataflow Graphs." University of Tokyo, Hongo Campus, Tokyo, Japan, December 1, 1992.

Arvind. "The Monsoon Dataflow Machine and its Performance." University of Tokyo, Hongo Campus, Tokyo, Japan, December 8, 1992.

Arvind. "The structure of the Id compiler." University of Tokyo, Hongo Campus, Tokyo, Japan, December 15, 1992; Fujitsu Labs, Kawasaki, Japan, February 4, 1993.

Brobst. "Object-oriented Development as a Core Technological Capability." Object World, San Francisco, June 15, 1993.

Brobst. "Resource Management in Massively Parallel Computing Architectures." Hewlett-Packard Laboratories, August 18, 1992; Boston University, December 2, 1992; National Television University, February 15, 1993.

Iyengar. "Dynamic Storage Allocation on a Multiprocessor." Daewoo Electronics Co., Ltd., Seoul, Korea, September 16, 1992; Department of Computer Engineering, Seoul National University, Seoul, Korea, September 18, 1992; Department of Computer Science and Engineering, Pohang Institute of Science and Technology, Pohang, Korea, September 22, 1992; Computer Architecture Section, Computer Science Division, Electrotechnical Laboratory, Tsukuba, Japan, September 24, 1992; Hewlett-Packard Company, Chelmsford, MA, October 23, 1992.

Joerg and Kuszmaul. "A Tightly Coupled Processor-Network Interface." Thinking Machines Corporation, September 9, 1992.

Kuszmaul. “A Tightly Coupled Processor-Network Interface.” Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, October 13, 1992.

Kuszmaul. “Performance Evaluation of Network Interfaces.” 1992 MIT Student Workshop on VLSI and Parallel Systems, July 21, 1992.

Papadopoulos. “*T: Building Blocks for Multithreaded Multiprocessors.” 40th Anniversary of SIAM, Los Angeles, CA, July 24, 1992.

Papadopoulos. “Machine Independent Machine Code.” DARPA PI Meeting, Orlando, FL, September 24, 1992.

Papadopoulos. “*T Architecture.” Presentation for Dr. Gary Koobs, Office of Naval Research, Cambridge, MA, October 16, 1992.

Papadopoulos. “*T: Multithreaded Building Blocks for Parallel Machines.” Siemens Corporate Research, NJ, November 2, 1992.

Papadopoulos. Supercomputing 92, Multithreading Computers Minisymposium, Panelist, Minneapolis, MN, November 20, 1992.

Papadopoulos. “Parallel Architectures for Fluid Flow Simulation.” Symposium on High-Performance Computing for Flight Vehicles, Grid Generation and Flow Simulation on Parallel Machines Session, Washington, DC, December 9, 1992.

Papadopoulos. “Taking RISCs.” University of California, Berkeley, CA, January 7, 1993; Hewlett-Packard, January 8, 1993; Motorola, Austin, TX, January 14, 1993; France Telcom (at MIT), Cambridge, MA, January 29, 1993; University of Connecticut, Storr, CT, February 19, 1993; McGill University, Montreal, Quebec, Canada, March 5, 1993.

Papadopoulos. Panelist: ”The Commercial Prospects for Dataflow Architectures.” IFIP WG10.3 meeting, Orlando, FL, January 20, 1993.

Papadopoulos. “Continuous Computing: The Fusion of Parallel and Distributed Systems.” University of California, Berkeley, CA, February 22-24, 1993.

Papadopoulos. ”*T: Scalable Building Blocks for Parallel Processing.” ARPA PI Meeting, Arlington, VA, April 14-16, 1993.

Papadopoulos. “Lattice Gas Methods.” Invited talk at Et Kappa Nu induction dinner, Cambridge, MA, April 23, 1993.

Papadopoulos. Panelist on Scalability in Distributed Systems Panel, The 13th International Conference on Distributed Computing Systems, Pittsburgh, PA, May 27, 1993.

Papadopoulos. “If Computers are Commodities, Who Needs Special Purpose Hardware?” Invited talk for MIT Visiting Committee, March 16, 1993.

Shaw. “Performance of Data-Parallel Primitives on the EM-4 Dataflow Parallel Supercomputer.” Frontiers ’92: The 4th Symposium on the Frontiers of Massively Parallel Computation, McLean, VA, October 19, 1992.

# Contents