LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Computation Structures Group Progress Report 1991-92

Computation Structures Group Memo 349
July 30, 1992

G.A. Boughton (ed.) and Yuli Zhou (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 N00014-89-J-1988.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Computation Structures Group

July 1, 1991 — June 30, 1992

Academic Staff

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

Research Staff

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

Graduate Students

S. Aditya	A. Caro	M. Flaster	J. E. Hicks	M. Sharma
B. S. Ang	Y. Chery	S. Glim	A. K. Iyengar	A. Shaw
P. S. Barth	K. C. Cho	M. Heytens	C. F. Joerg	
S. A. Brobst	D. Chiou	D. S. Henry	B. C. Kuszmaul	

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

A. M. Maderer
S. Hardy

Visitors and Adjunct Members

Z. Ariola	(Harvard University)
K. D. Chung	(Pu San National University, Korea)
C. Fournet	(Ecole Polytechnique, France)
D. Hwang	(Sung Kyun Kwan University, Korea)
M. Halbherr	(ETHZ, Switzerland)
M. Motomura	(NEC, Japan)
S. Sakai	(Electrotechnical Laboratory, Japan)
T. Senta	(NEC, Japan)
M. Srinivasan	(Indian Institute of Science, Bangalore)
S. D. Youn	(Pusan National University, Korea)

Computation Structures Group

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.

The 16 node Monsoon system at MIT has been operational since the Summer of 1991 and has proved to be fairly reliable. Motorola delivered a second 16 node Monsoon system to Los Alamos National Laboratory early in 1992, which came up quickly and has been working reliably ever since.

We reported last year on the development of the *T architecture. The detailed design of a hardware implementation for *T has started this year. The proposed overall structure of a physical *T node contains two 88110MP's, two memory controller chips (MC's), a network router, and DRAM. The detailed design is being done at Motorola, who is responsible for the 88110MP, and MIT, who is responsible for the MC and the network router.

Our work has continued on the theoretical study of languages able to capture the concept of sharing. Such languages can be seen as examples of a system called Graph Rewriting System (GRS), containing a block construct which describes precisely the sharing of subexpressions done by most interpreters of functional languages.

A theoretical framework based on abstract interpretation has been developed for defining the lifetimes of aggregate objects allocated by programs. An algorithm is developed based on the theory, which was implemented as a module of the Id compiler to verify or insert storage reclamation code in Id programs, with very good result.

Several frame managers are implemented for Monsoon that permit our applications to run well on both one processor and multiprocessor configurations. We have performed simulation experiments in order to compare several different dynamic storage allocators. Based on the experimental results three new algorithms were implemented which achieve better performance when the percentage of requests for large blocks is high.

We implemented the Monsoon I/O system containing a core of low-level routines providing flexible, high performance, parallel I/O operations. Several low-level I/O libraries were built on top of the core, as well as a high-level Id library of I/O functions.

In collaboration with Motorola, we have enhanced the capabilities of Monsoon Id World this year. These enhancements have enabled us to make the initial release of the system for external distribution. The release contains a new interface EZID or "Easy Id", which supports evaluation of Id expressions, manipulation of statistics, and interaction with MINT or the Monsoon hardware using an extension of Id 90.1.

The Id compiler in Id project made significant progress during the past year, and we are expecting to see a preliminary compiler generating code for TTDA before the summer ends. The goal of the project is two-fold: on one hand the compiler will provide the largest application written in Id, exercising all parts of the language, including the non-functional extension of I-struction and M-structures; on the other hand we expect this compiler will ultimately become the compiler for Id running on various hardware platforms. Our experiences with writing various compiler modules has been very rewarding, especially in dealing with M-structures where synchronization and parallelism has to be considered very carefully.

2 Personnel and Visitors

Arvind was honored with the Charles W. and Jennifer C. Johnson Professorship of Computer Science and Engineering.

Kidong Chung visited the group until September 1991. Chung was studying dataflow languages and architectures and resource management schemes.

Cédric Fournet has been visiting the group since April 1992 and will be with us until July 1992.

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

Daejoon Hwang visited the group until July 1991. Hwang was studying dataflow architectures.

Joanna Kulik received the Johnson Award for Outstanding Bachelor's Thesis, June 1992.

Masato Motomura has been visiting the group since August 1991 and will be with us until December 1992. He is a fellow with the Center for Advanced Engineering Studies.

Greg Papadopoulos received the Class of 1922 Career Development Professorship of Electrical Engineering and Computer Science.

Shuichi Sakai visited the group until April 1992. Sakai studied synchronization mechanisms for massively parallel computer systems. In September 1991, he received the IBM Japan Award.

Kishore Sakharkar was a UNDP fellow, sponsored by the University of Bristol and the Center for Advanced Engineering Study. He visited the group in January - February 1992.

Tetsuhide Senta visited the group until August 1991, researching memory management schemes for the *T processor.

Mandyam Srinivasan visited the group until October 1991, studying both the evaluation of parallel architectures and hardware design methodologies.

Sung Dae Youn visited the group from December 1991 - February 1992.

3 MIT-Motorola collaboration on Id, Monsoon, and *T

Our collaboration with the Motorola Cambridge Research Center (MCRC) and the Motorola Computer Group in Tempe, Arizona has been both intense and extremely productive. The three year contract 1989-1991 was successfully concluded. During the past year, Motorola completed engineering revisions to all Monsoon prototypes, delivered to Los Alamos National Laboratories an additional 16-node Monsoon consisting of 8 PE's and 8 IS's, an interconnection butterfly packet network, and appropriate enclosure with cooling and power suppliers. The Los Alamos machine is in their Advanced Computer Laboratory and has been up and running since the beginning of March. It is being used by a small group of Los Alamos computational scientists (the principle collaborator is Dr. Olaf Lubek) to try and understand how dataflow can affect solutions to scientific problems that, thus far, have been particularly intractable to speed-up by parallel computation. The Los Alamos machine is available over the network for experimental purposes to other scientists at universities and research laboratories.

Motorola has manufactured six additional 2-node Monsoon systems (additional to the number specified in the contract) which the Laboratory along with DARPA's advice will place as experimental prototypes in the dataflow research community to encourage experimentation and software development.

In addition to completing our Monsoon collaboration, scientists and engineers from the Laboratory and Motorola have had an intense dialog concerning the new *T architecture and its design and the division of labor and responsibility between the Laboratory and Motorola for the new project. These discussions have been very fruitful as a *T architecture has been specified and a high level design completed (as explained in following sections).

Motorola had an exhibit at last November's Supercomputer Conference featuring Id/Monsoon and relevant work done by the Laboratory, Motorola, Los Alamos National Laboratories (Lubeck), Sandia National Laboratories (Hoch), and the University of California at Berkeley (Culler). In addition to a 14 minute video about Dataflow and Monsoon, made by Motorola, which ran continuously in the booth, members of CSG had a strong representation at the conference. The video featured Professors Arvind and Papadopoulos of the Laboratory and Dr. Traub of MCRC.

During the year, we have held numerous informal meetings with MCRC and Tempe personnel often weekly or a more frequent basis when needed. In addition, the following formal meetings were held:

July 22, 1991:	Joint review of Monsoon and preliminary planning for *T.
September 18-22, 1991:	Supercomputer Conference, Albuquerque, NM.
September 25, 1991:	Motorola Corporate Research, Cambridge, MA.
December 2, 1991:	Joint review of Monsoon and *T activities, Cambridge, MA.
February 11, 1992:	Joint review of Monsoon and *T activities, Cambridge, MA.
June 9, 1992:	Joint review of Monsoon and *T activities, Cambridge, MA.

4 Other external collaborations

Our research continues to benefit from the collaborative efforts of our group 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. This community of collaborators will expand with the public release of Monsoon Id World.

4.1 DEC Cambridge Research Lab

Rishiyur Nikhil has continued his collaboration with the group. He has worked with Professor Arvind on the preparation of a text book on the programming language Id. He has worked with Professor Arvind and Professor Papadopoulos on issues in Id language design, Id compilation, and multithreaded architectures.

4.2 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 are investigating using Culler's backend for the Id Compiler on our Id Compiler in Id project for execution on stock hardware.

4.3 Sandia

Sandia is continuing its work in threaded dataflow computing. In addition to their Epsilon-2 processor and system development, work on the migration of applications to Id on Monsoon progresses. Sandia has developed a version of a low density fluid dynamics application called DSMC (Direct Simulation Monte Carlo) for execution on the Monsoon system. DSMC is used extensively at Sandia to model the aerodynamic and thermodynamic behavior of reentry vehicles in the upper reaches of the earth's atmosphere. It is also beginning to attract attention from the semiconductor industry as a method of simulating advanced semiconductor processes.

Sandia has coded DSMC in Id and executed it on a single processor Monsoon system. The application, complete with graphics, was demonstrated in the Motorola booth at the Supercomputing '91 conference. Future plans for DSMC on Monsoon focus on two goals: executing the program on a multi-processor Monsoon system and augmenting the program with more complex physical models.

4.4 Los Alamos

A team at Los Alamos, headed by Olaf Lubek, has continued its effort to investigate the feasibility of the dataflow model of computation and functional languages designed for numerical computation. Development of application codes provides performance and scalability data and drives further development. Towards this end, we have selected a very general Monte Carlo transport code called MCNP as a simulation that can benefit significantly from a massively parallel MIMD architecture.

Progress on a model MCNP code for Monsoon

The MCNP code has many applications in diverse fields such as nuclear reactor safety, medical dosimetry and oil well logging. We are redeveloping this application in Id. Our intention is not to rewrite MCNP in its entirety because it includes extensive user features that are unimportant to its performance, but rather the core of the code that retains its numerical and physical complexities (we will refer to this code as MCNP-ID). During the past year, we have developed the following capabilities in the code:

- a general user-specified geometry
- simple and detailed photonics transport
- numerous variance reduction techniques
- statistical tally information

Currently, we are executing MCNP-ID on our 16-node Monsoon machine solving benchmark problems with known solutions. This is the first time that we have had enough computer power to execute real-world problems for an Id application. This computational power will allow us to execute large enough problems to get statistically valid answers. In the future, a direct comparison between the Fortran MCNP and MCNP-ID will be possible.

4.5 Colorado State University

During the summers of 1990 and 1991, Bob Hiromoto (LANL) and Wim Bohm (CSU) worked on algorithm design for dataflow execution. This collaboration was funded by Los Alamos National Laboratory. Our approach aims to evaluate the expressiveness of functional languages and the efficiency of both the compiler and its supporting parallel hardware.

The methodology encompasses the analysis, design and implementation of numerical algorithms written in a functional style. As Id also allows for non-functional styles of programming, we have the opportunity to evaluate functional and non-functional solutions to certain problems in one language framework. Since it is our research interest to assess the generality of the dataflow computational model for numerical applications, we have chosen to study typical Fortran library routines that form the core of a numerical analyst's tool kit. The advantages of such numerical routines are the availability of algorithm documentation, the relatively concise policy of memory management as expressed in Fortran, the ease of analysis, and most importantly the complexity of their computational and data structures. A number of routines have been written in Id and targeted for the Motorola Monsoon Dataflow Machine. We have used the dataflow oriented complexity measures *total work*: the total number of instructions executed, and *critical path length*: the number of time steps required to execute the program if an infinite number of processors were available. We have studied the dataflow performance of our algorithms under the parallel profiling simulator Id World. This approach has allowed us to follow the computational and structural details of the parallel algorithms as implemented on dataflow systems.

We have examined problems that exhibit different computational characteristics. We have designed algorithms for the Fast Fourier Transform which exhibits an interesting computational parallelism with data dependences between array elements in the various butterfly shuffles. We have started work on adaptive quadratures, where the problem is to control the dynamic unrolling of recursively adaptive grid refinements. We have designed a parallel Jacobi eigenvalue/vector Solver, and recently completed a Householder-QR eigensolver. Some of our results were presented in the Dataflow session of the 1991 IMACS conference [6] and in a presentation at MIT, August 16, 1991. Recently, we have submitted a paper to the ISCA Dataflow Workshop [5] comparing recursive and iterative versions of Fast Fourier Transforms in Id, and we are currently preparing a paper on resource usage of Id programs for the Special Issue on Dataflow and Multithreaded Architectures of JPDC.

We are waiting for the final decision of Motorola to fund further research in this area.

5 Id: general topics

The Id language remains quite stable after the introduction of M-structure and explicit sequencing, which proved to be indispensable when programming realistic system programs such as the Id compiler in Id.

5.1 Graph Rewriting Systems: capturing sharing of computation in language implementations

Zena Ariola and Arvind have continued to work on the theoretical aspects of the two intermediate languages Kid and P-TAC. In particular, Kid and P-TAC can be seen as examples of a system called Graph Rewriting System (GRS). The basic feature of a GRS is the *block* construct, *i.e.,* letrec, which is a first class term and not merely syntactic sugar for function application. Each subexpression in a GRS is given a unique name and only "*values*" and "*names*" can be substituted freely. We believe that GRSs are suitable to describe precisely the sharing of subexpressions done by most interpreters of functional languages. As shown in [1] GRSs are also useful to describe the operational semantics of a wider class of languages than pure functional languages.

Much of the past work on graph rewriting has been to prove its correctness with respect to either the λ-calculus or Term Rewriting Systems (TRSs), while our interest in graph rewriting is more general. We see graph rewriting as a system in its own right, and want to explore its syntactic and semantic properties. In particular, we see GRSs as a suitable formalism to express side-effect operations. We also want to include graphs with cycles and as well as rules that recognize or create cycles. This is not the case in either [15] or [4] where only acyclic graphs are considered and thus, some important implementation ideas are ruled out. In [2] we introduce GRSs and prove many syntactic properties of such graph rewriting systems.

We develop a term model a la Lévy [12] for GRSs and prove some of its semantic properties. We restrict our attention to GRSs which are adequate to describe sharing in combinatory systems.

We introduce a notion of *instant semantics* [16] associated to a term. The instant semantics captures the stable information contained in a term without executing it. We then collect all the information gathered by reducing a term in a set which represents the *information content* of a term. We then show that in the absence of interfering rules the information content defines an interpretation function, *i.e.,* it is sound with respect to reduction and it defines a congruence relation over the set of terms.

We use the information content as our criteria for proving the correctness of optimizations. To that end we define a syntactic ordering on the set of terms based on the amount of sharing contained in a term. We are then able to show that if a term M has less sharing than a term N, then the information content of M is less than the information content of N. Due to the properties of the information content we are guaranteed that for any context $C[\Box]$, the information of $C[M]$ will still be less than the information of $C[N]$. We can characterize the effect of some optimizations, such as the common subexpression elimination, the lifting of free expression and invariants from a procedure or loop body, as increasing the amount of sharing in a term. Therefore, in the absence of interfering rules, according to the previous result these optimizations are partially correct.

We also explore in [3] under which conditions the amount of sharing does not affect the information in a term. To that end, we discard sharing from our observations and we introduce the notion of answer of a term. We show that in the absence of non left-linear and left-cyclic rules the answer is a congruence. Consequently, we have that under those restrictions if a term M has less sharing than a term N then no context will distinguish between M and N. In other words, the common subexpression elimination, and the lifting of free expression and invariants from a procedure or loop body are totally correct. Since the notion of answer is not tied up to the termination property of a term, we are also able to show that the cyclic implementation of the Y-rule is totally correct.

We finally explore in [3] the correspondence between graph rewriting and term rewriting. We show that acyclic orthogonal GRSs, *i.e.,* GRSs without interfering rules, non left-linear rules and left-cyclic rules, are a sound and complete implementation of term rewriting. In this regard, the approach taken by other researchers [7, 11] is based on infinite rewriting. On the other hand, our approach is based on showing that the behavior of a graph can be deduced from its finite approximations. In other words, we show that graph rewriting is a "continuous" operation.

In future we plan to work on incorporating the λ-calculus and I-structures in our model.

5.2 Partial Evaluation

The field of *partial evaluation* (PE) has burgeoned in the last few years with a spate of conferences and papers indicating increased interest by the research community. Two reasons seem apparent for recent developments. First, language developers have had their eye on PE techniques as a way of automatically generating compilers from interpreters. Secondly, compiler writers see PE (or more properly called *procedure specialization* (PS) in this domain) as a natural extension of the constant propagation code transformation. These two views of PE are merely two ways of looking at the same problem, and not mutually inconsistent. However, choosing a particular view colors one's methodology and one's measure of success or failure. For example, efficient self-applicable partial evaluators are important for compiler generation, but of less interest when seen as a special case of PS. Accordingly, in research on PS we're taking "the road less travelled" and examining the consequences of looking at PS as an extension of the constant propagation transformation.

Steve Glim is investigating extensions to the id compiler that incorporate different forms of procedure specialization to produce code that uses resources more efficiently. He is especially interested in language transformations that lessen the amount of intermediate storage required. Also, he is investigating transformations that efficiently specialize procedures on partially specified data and incorporate higher order functions in the specialized code.

5.3 Basic Input/Output

The Monsoon Input/Output System follows a layered design approach. This section discusses the core of the system, a substrate of low-level routines that implement the basic I/O capabilities of Monsoon. The main goal of the substrate is to provide a flexible, high performance, parallel set of building blocks on which to base the implementation of more complicated I/O systems (as discussed in later sections).

At its innermost, the I/O substrate implements a highly efficient and highly parallel set of transport routines that shuttle data between the Monsoon processor and its front-end Unix host. The transport routines are based on the notion of an *I/O channel*, a data-structure that is used to *virtualize* the I/O resources of a particular machine configuration. I/O channels allow the data transport routines to operate robustly even in the face of different machine configurations that may arise due to processor, memory controller, or network switch failures.

Basic Input/Output libraries

Several low-level I/O libraries have been built directly on top of the transport substrate. These provide the support for the different services of the I/O system. A brief description each library follows:

File Library : implements block read/write routines to binary data files. These routines are fast, since block data transfers are utilized, and flexible, since they operate on unformatted binary data.

Console Library : implements a simple terminal interface utilizing the capabilities of the standard `xterm` terminal emulator. For convenience, VT100 control sequences are implemented on the output side of the interface, while a simple line editor is implemented on the input side of the interface.

O/S Interface Library : implements a basic interface to the services of the front-end operating system. Currently, these services include the gathering of I/O operation profiles.

Experimental Graphics Library : implements a basic rendering interface to the X Window System. Currently, this library includes operations to display bitmap images; in the future, a complete set of graphics operations will be implemented.

Use and performance

The basic Monsoon I/O System has been used in several programs, most notably the SPLASH benchmarks ported by Michael Tso and the DSMC Low-Density Fluid Dynamics Simulation written by James Hoch of Sandia National Labs. In both cases, the file I/O subsystem was used to read in large initial data sets in binary form. In a separate effort, the graphics I/O subsystem was used by Michael Davidson to implement a ray-tracing application for Monsoon.

In the few applications mentioned above, the performance of the I/O system has been satisfactory but not overwhelming. Careful tuning over the next year should lead to significant performance increases across the whole range of I/O services.

5.4 High level Input/Output library

Christine Flood has implemented an ID library of i/o operations for monsoon. The functional specification for this library was developed by David Culler's group at Berkeley and is referred to as the Berkeley sequential IO proposal. The implementation of this library is called stdio.

The library provides three basic stream types: file streams, string streams, and console streams. All stream types are buffered for efficiency. Id functions are provided for formatting and scanning common Id data types such as integers, floats, booleans, lists, strings, arrays, etc to/from these streams.

Sequentialization

In a parallel language the order in which I/O statements are executed is unpredictable. We want to be able to output objects in a deterministic order.

There are currently two methods of sequentializing operations in ID. The Berkeley I/O proposal uses explicit barriers, so two format statements would look like:

```
{ _ = format e1;
  ---
  _ = format e2 };
```

Using the MIT threaded I/O library that we are currently working on, two format statements have the form:

```
{ s1 = thio_format e1 trigger;
  _ = thio_format e2 s1 };
```

which depends on explicit triggers to sequentialize I/O operations.

It is expected that the MIT library will have more parallelism because of less unnecessary serialization. The value of `e2` may be calculated while the value of `e1` is being printed

Parallel Input/Output

In the future we plan to experiment with our sequential I/O libraries to determine appropriate methods of implementing parallel I/O. Work has begun on a library for reading/writing large arrays quickly in parallel.

6 Id: compiler and run-time systems for Monsoon

The Id compiler for Monsoon has improved substantially due to many enhancements in the mid-end optimization modules and new, more efficient schemas in the back-end code generation modules.

6.1 Storage reclamation in Id

James Hicks completed doctoral work on compiler-directed storage reclamation. He developed a theoretical framework based on abstract interpretation for defining the lifetimes of aggregate objects allocated by programs. He developed an algorithm based on this theory and added a module to the Id compiler that implements this algorithm. This module enables a version of the Id compiler to verify or insert storage reclamation code in Id programs. The modified compiler is quite effective at determining where it is safe to deallocate storage in scientific programs. The more regular the control and data flow are in the program, the more likely the compiler will be able to determine object lifetimes. The compiler is less effective on programs using recursive types such as trees or on programs that have large amounts of sharing of data structures.

The analysis framework is based on an operational semantics of Id. The operational semantics is defined by a store-based interpreter, even for the functional subset of the language. The store maps object labels to object values, and all references to aggregate objects indirect through the store. This indirection is necessary in order to model object sharing correctly. Evaluation of an expression by the interpreter yields a value, which is either a number, a boolean or an object reference, and a store, where objects are defined. In addition, the interpreter computes the set of labels of objects that were allocated, dereferenced and deallocated during the evaluation of a subexpression.

We can use the interpreter to tell us exactly which objects are allocated by an expression, and which are reachable from a given value with respect to a given store. From this we can determine the exact lifetimes of objects during a particular execution of a program. If an object is allocated during the evaluation of a subexpression and that object is not reachable from the result of the subexpression, then its lifetime is contained by the lifetime of the subexpression. We can use this information to determine exactly when it is safe to deallocate an object during a particular execution of a program.

However, in order to determine statically that it is safe to deallocate the object bound to a particular variable, we must show that under every execution of the program, and for any input data, that it is safe to deallocate the object. For this, we *abstract* the operational semantics to yield an interpreter that gives us a *summary* of the behavior of a program over all executions. Likewise, we generalize the criteria we use to determine the lifetimes of objects.

The algorithm used to analyze a program and to determine where it is safe to deallocate objects is essentially an abstract interpreter to generate a table of input-output mappings that show the behavior of each function, and then to use these mappings while analyzing the body of each function.

This work has resulted in solid theory for some of the work we did on storage reclamation annotations in the past few years. We have also extended the compiler so that it can deallocate objects in more cases than it could previously by adding conditional deallocation code that tests for sharing at run-time before deallocating an object, in cases when the compiler cannot determine statically that there is no sharing. This case arises in particular in loops that may execute zero or more times and that allocate an object on each iteration.

This framework models Id with functional tuples, arrays, and algebraic types. Because the interpreter is store-based, we can also model single-assignment elements of I-structure arrays and algebraic types. We also have an abstract semantics for M-structure arrays and algebraic types. The implementation of this analyzer/transformer has been very successful when applied to scientific codes. The deallocation code added to the Simple, Gamteb and Wavefront benchmarks reclaims all of the intermediate storage allocated by those programs. In addition, we have implemented a limited form of sharing analysis that allows the compiler to insert code to deallocate the intermediate arrays of tuples in an implementation of the FFT algorithm.

The compiler is not as successful with programs that use recursive types allocated by recursive procedures, because it does not have a strong enough sharing analysis to determine if an object is a tree, an acyclic graph or a cyclic graph — it has to assume it is a cyclic graph. This is an area we intend to explore further.

Another drawback is that the analysis is very expensive. Although the algorithm to insert or verify deallocation commands is fairly efficient once the input-output behavior has been determined for each procedure, the process of computing input-output behavior takes a tremendous amount of time. Compilation times have increased by a factor of two to a factor of ten when the analysis and transformations are done. This is not the kind of thing that can be installed in a development mode compiler, but it's not unreasonable to use once a program is debugged and is ready to go into production runs.

6.2 Efficient Implementation of Loops

Boon Ang has been working on improving the implementation of loops on Monsoon. One area of focus is sequential loops which usually form the innermost loops of programs and have the greatest impact on execution time. Sequential loops are tricky to implement because they reuse frame storage and the execution within each iteration proceeds in parallel. Our old sequential loop implementation, while correct, was not very efficient. Two new sequential loop schema were developed and implemented in the Id Compiler's Monsoon Backend. These requires gathering some strictness information from the code. The second scheme exploits the fact that the loop is sequential and try to push all synchronization to the loop iteration boundary. The extent to which this is possible and result in savings in the number of cycles executed is somewhat restricted by the architecture. Nevertheless, both new schemas turn out to be substantially more efficient then the old schema and are used by default in the production verion of the Id compiler.

The set-up and clean-up phases of k-bounded loops were re-implemented by Boon. They now proceed in a recursive, parallel fashion on all the frames involved and hence has a shorter critical path. The old set-up and clean-up phases were each performed by a sequential loop and was thus sequential. The new approach also takes fewer total instructions in addition to reducing the critical path.

Automactic strip mining by the compiler was implemented in January. This transforms certain k-bounded loops into a doubly nested loop, with the outter loop executing in parallel and the inner one sequential. The original loop body is now the loop-body of the inner loop. Where strip mining is applicable, it has two big advantages:

- It allows us to use the sequential loop schema, which is much cheaper then the k-bounded loop schema, while offering parallel execution of the loop.
- It breaks the coupling between the k frames of a loop, allowing each to proceed independently at full speed. Previously, if one of the k frames resided on a PE that was busy executing other work, the entire loop stalled. With strip mining, once the k sequential loops are started, a frame on a busy PE will not affect the execution on the other $(k-1)$ frames.

Finally, Boon also implemented lifting the initialization and cleanup of loops within a loop nest as far out as possible. This turned out to be tricky and complicated because the initialization may be lifted through several k-bounded loops. So far, of the 5 benchmarks that we run regularly, it has proven to be useful in only one of them, SIMPLE, where it reduced run time by 10%. However, this optimization introduces substantial complexity to the compiler. We are studying it's effects on more programs in order to decided if we want to include it in the production version of the compiler.

6.3 Efficient manipulation of literal values

Stephen Glim has studied the efficient manipulation of literal values. A literal value is a wholly known value introduced by the programmer in his or her program. The final value of any literal is independent of any of a program's inputs and (as opposed to an input variable), will have the same value during every execution of the program. Intermediate computed variables are derived by combining literals and input variables (as well as other intermediate values). Intermediate variables that depend only on literals might be literals themselves (if the computation computing them halts) and compile time optimization may be able to compute these "intermediate" literals before the program executes.

A language may supply syntactic support for structured literals by having special syntax that its parser can recognize as denoting a structure built entirely of literals. Failing that, a language may provide general purpose syntax for simply specifying structures, which need not contain only literals, but makes them easily specified. Id takes the latter course by providing syntax that makes structured constructs easy to express, but not constraining them to be wholly composed of literals.

However, the question remains, how does the compiler take these simplified notations for structures and translate them to executable code? The simplest way to translate this notation is to "desugar" it into program fragments of simpler structure allocators and assignments. This solution has the admirable property of machine independence, since complicated syntax is desugared into simpler (though more verbose) programs that can be directly translated.

It's clear that *large* literal structures will significantly degrade the performance of a machine independent desugaring approach to their specification. In desugaring, every literal produces a fragment of program text proportional to the literal's size, which then becomes incorporated into the intermediate data structure. Both traversing old and building new intermediate structures may take time and resources proportional to the "size" of the intermediate structure. If a large literal structure (encompassing perhaps thousands of atomic literals) is specified then the intermediate form will grow by a similar size. Most painfully, optimization analysis and rebuilding may now need to traverse these newly generated statements and program compilation will take great resources of time and space.

None of this corresponds to the programmer's model of how the language should be compiled. Literal structures, are by definition already optimized (excluding garbage detection) and need not be continuously traversed by the compiler but merely collected together and loaded into memory at execution time. Compiling methodologies that depend on desugaring to compile literal structures will defeat this intuition about the complexity of compilation and make programs that seem simple (but specify large structures of literals) inefficient to compile.

The solution to all these problems is to not naively desugar literal structures into program fragments, but instead apply a more specialized strategy for handling them when they appear. It's imperative that literal structure's get represented by a structure accessible in constant time and represented in constant space (wrt the optimizer) in the intermediate form, otherwise there can be no benefit during the optimization phases of compilation.

Current development work includes extending the id compiler with more support for structured literals. Literals will be detected early and manipulated efficiently by the compiler before being passed to the loader special objects to be loaded directly, rather than constructed during program execution

6.4 Frame manager

Derek Chiou has continued his work on frame managers for Monsoon.

In a sequential computer, stack frames are allocated and deallocated by manipulating a stack pointer. A parallel computer's frame manager is more complicated since more than one frame might be active at any time, and the ordering of allocations and deallocations is much less strict. This past year we have implemented several frame managers that permit our applications to run well on both one processor and multiprocessor configurations.

Our current default frame manager distributes load using a simple round-robin scheme that has been able to balance loads extremely well. We have seen linear speedup for a few programs and very close to linear speedup for most of our other applications. The frame manager is written in MONASM, our assembly language. So far, the default frame manager has been running for several months and seems very robust. Current research is being done on coalecing frame managers.

6.5 Heap manager

Arun Iyengar is studying ways to efficiently manage heap space (also known as dynamic storage) on multiprocessors. Heap objects may have indefinite sizes or indefinite lifetimes. The run-time system allocates and deallocates heap objects.

We have performed simulation experiments in order to compare several different dynamic storage allocators. Algorithms similar to quick fit achieve the best performance in both parallel and sequential environments. Quick fit achieves very fast allocation by using separate free lists for blocks of different sizes. Mr. Iyengar's Ph.D. thesis explores methods of improving upon the performance of quick fit by modifying the manner in which large free blocks are stored.

We have developed three new algorithms which achieve better performance than quick fit when the percentage of requests for large blocks is high. Quick fit stores all large free blocks on a single linked list. By contrast, *multiple free list fit I* uses several linked lists for large blocks. *Modified quick fit* stores all large blocks in a B-tree. *Multiple free list fit II* uses several lists and a B-tree for storing large blocks.

Storage managers utilizing both quick fit and multiple free list fit I have been implemented on Monsoon. High throughputs are achieved by utilizing multiple free lists which can be searched concurrently. The storage manager is designed so that throughput scales linearly with the number of processors.

6.6 Id World

In collaboration with Motorola, R. Paul Johnson has enhanced the capabilities of Monsoon Id World this year. These enhancements have enabled us to make the initial release of the system for external distribution. Specifically these include:

- automated startup, initialization and shutdown of MINT and MHD, the Monsoon Hardware Driver
- session save and restore procedures
- statistics and colored code block statistics collection and display
- EZID, an Id language command and expression listener

Monsoon Id World 1.1

Monsoon Id World V1.1 was released in May for external distribution. The distribution includes all the Monsoon client programs, a prebuilt Id World lisp image and the MINT interpreter. The Monsoon Id World is a "shrinkwrap" agreement which is activated upon receipt of the system. The Id World lisp image is built on Lucid's SPARC Application Environment. As with GITA Id World, there will be a one time charge for media and documentation. We are planning to port Id World to the IBM RS6000.

EZID

EZID or "Easy Id" significantly simplies the interface to Monsoon Id World. Under EZID, users interact with Id World; evaluating Id expressions, manipulate statistics, and interact with MINT or the Monsoon hardware using an extension of Id 90.1. EZID interfaces to the compiler and loader for compilation and evaluation of complicated expressions. New users have found this interface to be intuitive.

6.7 Id compiler in Id

The Id compiler in Id project was started in April 1991, with the goal of providing a large application in Id, as well as becoming the future compiler for Id on many hardware platforms. Major coding started in the summer of 1991, towards the end of that summer we had a preliminary front-end tested on some small example programs.

During the fall of 1991, a lot of efforts were spent on consolidating the frontend. Using Berkeley Yacc, Yuli Zhou generated a real parser for Id, and implemented pretty-printers for the Compiler's intermediate representations. Shail Aditya debugged the type-checker (written by Sigbjorn Finne and Roy Seto), and integrated the system to compile a number of example programs including the Paraffins, which is a real-sized Id program.

In spring 1992 we decided to go ahead with the remaining stages of the compiler: a mid-end doing optimizations on a Graph representation of Kid (Kernel Id) and PTAC (Parallel Three-address Code), and a back-end generating code for the TTDA architecture. Joanna Kulik, with the help of Shail Aditya, designed and implemented the Kid-graph as well as a number of optimizations on Kid-graph as her Bachelor's thesis. Yuli Zhou implemented a preliminary version of a PTAC to TTDA-graph translator, which was able to generate code for some small programs that actually run on GITA. Towards the end of spring Cédric Fournet, visiting from Ecole Polytechnique of France, undertook the project of implementing Kid to PTAC translation. By that time the general structure of the compiler was well-understood. The compiler has 5 internal representations: Aid (abstract Id), Kid, Kid-graph, PTAC and TTDA-graph. The functional modules of the compiler are described in the following, which are stream-lined to form the compiler:

Parsing The parser constructs a parse-tree in Aid from an Id source program, where the nodes contain line/column number annotations indicating their corresponding positions in the source file.

Desugaring A preliminary transformation of the Aid parse-tree removes some syntactic sugar from the Aid program: List and array comprehensions are translated into nested loops; multi-clause definitions and functions are translated into simple ones with a case-expression in the body.

Scope-Analysis The main task of scope-analysis is to relate uses of variables to their definitions according to lexical scoping. A record is created at this time for each variable, uses of the variable will share this record through pointers. The collection of all such records constitutes what is commonly known as the symbol table. Many properties are gathered for declared variables into the records.

Translation to Kid Kid is a much smaller language in which temporary variables are introduced for intermediate values of expressions. Kid only has simple case-expressions without pattern-matching, thus the major task of Aid to Kid translation is to translate pattern-matched constructs into simple selections and nested simple case-expressions.

Type-Checking Being a functional language, Id has a Hendley-Milner style type-checker with extensions to deal with I-structures and M-structures. The type-checker also supports resolution of overloaded identifiers, which can be user-declared. Modules up to this one constitutes the so-called front-end of the compiler.

Translation to Kid-graph The Kid which the type-checker works on is still a tree-like structure, where sharing only occurs on variable records. It is a representation suitable for the type-checker, but not for various optimizations that would like to see the sharing of expressions represented directly. We thus translate Kid into a graph representation. A Kid-graph can be viewed as a tree of blocks, inside each block instruction can be shared in arbitrary manner. Block is a very important construct since it encapsulates the unit of control in a parallel language.

Optimizations on Kid-graph Many optimizations will be performed on Kid-graph, including constant propagation, fetch-elimination, common-subexpression elimination (CSE), call substitution and fast-call optimization, etc.

Translation to PTAC PTAC is similar to Kid-graph, except that in PTAC all data-structures and well as functional closures are explicitly represented using vectors of memory words. Thus the main task of Kid-graph to PTAC translation is to represent data-structures (vectors, records, arrays of various dimensions, algebraic types and closures) and to generate the correct function-call sequence.

Optimizations on PTAC All optimizations on Kid-graph can be performed on PTAC. Certain optimizations such as fetch-elimination and CSE will have the most visible effect since the previous translation introduces many redundant fetches and subexpressions due to the expansion of things like array indexing.

Signals and Triggers Signals and triggers are added to PTAC very late since their presence interferes with optimizations and are not needed by the optimizations. Modules working on Kid and PTAC graphs constitute the mid-end of the compiler, which are responsible for almost all of the optimizations.

Back-end for TTDA This includes a module to translate PTAC into TTDA-graph, a module to do some peep-hole optimizations and a final assembler module to write the object file in a format that can be loaded onto GITA.

As of now, Cédric Fournet implemented the key optimizations on PTAC, and Yuli Zhou is in the process of updating the back-end for TTDA. We are expecting to have a complete compiler able to compile some test programs in mid-July. The compiler itself, however, will by no means be completed at that time: the parser is not runing with much parallelism; pattern matching is not fully implemented due to an expected change in Kid and PTAC to accomodate sharing of blocks; overloading resolution is not yet implemented; and we do not yet have a full batch-compiler which can support separate compilation of files. Our UROP students for this summer have already chosen to work on some of these projects: Jan-Willem Maessen will implement overloading resolution, Laura Feeney started working on a complete implementation of pattern-matching based on decision-trees, Matthew Condell will try to propagate the line/column number annotation across various modules for the purpose of pin-pointing errors in the original source program, Roy Seto will implement loops in the PTAC to TTDA-graph translator.

Our decision to generate code for TTDA is based on the fact that it is the simplest target having a well supported environment (debugging, statistics collecting, etc). Our real interest, however, is in generating code for real machines such as Monsoon, *T, and even Unix workstations. Apart from requiring different back-end s, this needs sophisticated partitioning/threading analysis to generate bigger threads. These are the projects planned for the near future.

6.8 Test suites

Christine Flood developed an automated test suite to run the matrix multiply, gamteb, and wavefront example programs and verify their results. The tests were configured for 1,2,4, or 8 processing elements. These tests were used at MIT and Motorola to isolate hardware failures.

Christine developed an automated compiler test suite which verifies that every function in the reference manual works as documented.

R. Paul Johnson developed a regression test suite. As bugs were detected they were fixed and added to the test suite. This ensures that the same bugs don't reappear in future releases.

7 Monsoon hardware

The 16 node Monsoon system at MIT has been operational since the Summer of 1991. While we have experienced a few board level failures, the reliability of the hardware has been fairly good. We have not experienced any failures with the Monsoon network.

Over the year we have identified a few more hardware design issues and we have remedied them all with simple engineering changes (ECO's). These ECO's have been incorporated into all existing boards and will be incorporated into all future boards.

Motorola delivered a second 16 node Monsoon system to Los Alamos National Laboratory early in 1992. This system came up quickly and has worked reliably ever since.

8 Applications and performance measurements

We have continued to study a variety of Id applications and the level of performance that can be achieved for these applications on the Monsoon hardware.

8.1 Matrix Multiply, Gamteb, Simple and Paraffins

Boon Ang and Derek Chiou have continued to study the performance of these four Id applications on Monsoon. Their analysis has ultimately lead to a series of significant improvements in the Monsoon frame manager and the Monsoon backend of the Id compiler. Table 1 shows the improvement in run time of these programs over the past year and a half.

This study is still underprogress. We are now comparing these numbers to those of corresponding C or Fortran program running on MIPS microprocesses to study the efficieny of dataflow execution on Monsoon. We are also studying the speedup of these programs on the 8-PE 8-IS Monsoon. So far, these have shown that we are able to balance load fairly well and achieve good speedup in most cases.

Program	Feb, 91 (min:sec)	Aug, 91 (min:sec)	Mar, 92 (min:sec)	Jun, 92 (min:sec)
Matrix-Multiply (500x500)	4:04	3:58	3:55	1:48
Gamteb-9c (40,000 particles)	17:13	10:42	5:36	
Simple (grid size=100x100, 100 iterations)	0:19	0:15	0:10	0:06
Paraffins (n=19)	0:50	0:31		0:02.4

Table 1: Single Processor Performance

8.2 B-trees

Arun Iyengar is studying concurrent algorithms on B-trees. *Basic B-trees* contain data within internal nodes. All leaf nodes in a basic B-tree are empty. By contrast, B^+*-trees* store all data within leaf nodes.

We have implemented concurrent algorithms on both basic B-trees and B^+-trees in Id. Maximum concurrency on B^+-trees is obtained by using B-link algorithms. By contrast, B-link algorithms cannot be easily adapted to basic B-trees. We have obtained the best performance on basic B-trees by resorting to optimistic, top-down restructuring algorithms.

The performance of concurrent B-tree algorithms is highly dependent on the mechanisms which are available for allowing exclusive access to shared data. Our concurrent B-tree programs make extensive use of the imperative features of Id. M-structures and sequentialization constructs are prevalent.

8.3 Water

Water is an N-body molecular dynamics simulator, one of the Stanford Parallel Applications for Shared-Memory (SPLASH)[14] set of benchmarks. Water evaluates forces and energies in a system of water molecules in the liquid state. The computation is done over a user specified number of time-steps. Every time-step involves solving the Newtonian equations of motion for water molecules in a cubical box with periodic boundary conditions, using Gear's sixth-order predictor-corrector method[8]. The total potential is computed as the sum of intra and inter molecular potentials.

Michael Tso implemented two versions of Water in Id, one is non-functional (uses M-structures and barriers) while the other is functional (no explicit sequencing). The algorithm and data structures in both versions are the same as the C implementation in SPLASH. The functional version exploits more parallelism but incurs more overhead.

The following table gives details of several runs of the non-functional Id implementation of Water on a one processor configuration of Monsoon as compared to the C implementation on a MIPS workstation. The statistics that he was able to collect was restricted by bugs in the stats-mode software.

molecules	time-steps	Monsoon instr	MIPS instr	Monsoon CPI	MIPS CPI
8	2	4.93×10^6	3.12×10^6	1.73	1.04
27	2	5.92×10^7	1.15×10^7	1.32	1.16
64	2	3.35×10^8	4.86×10^7	1.24	1.20

Most of the extra overhead incurred by the Id program came from allocating heap objects. Although there are lots of opportunities to reduce the runtime overhead in Water, it will always be significant and underscores the price we must pay for our programming model. As the size of the data set increases, we observe that the C program's CPU utilization dropped due to cache misses. But the Monsoon processor does not have a cache and is able to exploit the extra data parallelism.

The multiprocessor configuration of Monsoon was temporarily out of service when statistics were collected on Water, thus no data is currently available on how well Water scales on Monsoon.

The program is approximately 1500 lines and is written entirely in Id. Most of the development work and debugging was done in EZID, on MINT or Monsoon hardware. Improvements were made to EZID and the I/O library during the course of Water's development. This software platform has become stable enough for developing relatively large applications.

9 *T hardware

We reported last year on the development of the *T architecture. As was discussed last year, the *T abstract model for a node contains three primary components; the Data Processor, the Start Processor, and the RMem processor.

The Data Processor is a conventional RISC microprocessor, and executes sequential threads supplied to it by the Start Processor. Whenever the Data Processor wishes to perform a remote access, *e.g.,* to load the contents of an address that is on another node, it issues a message to the remote node and *continues executing*. In particular, it does not wait for the response, which may arrive after considerable delay (long latency). Instead, the message that was issued, an `msg_rload` type of message, contains, in addition to the target address, a continuation that names a thread that must be scheduled when the response finally arrives.

Arriving at the remote node, the `msg_rload` message is processed by the RMem Processor, which sends the response in a `msg_start` message back to the original node. In particular, note that the remote node's Data Processor is completely undisturbed by this action. The `msg_start` message contains the value read from memory as well as the continuation information that came in on the `msg_rload` message.

When the `msg_start` response arrives at the original node, the Start Processor stores it in the node's memory and, using the continuation that rode piggyback on the message, schedules and feeds the named thread to the Data Processor.

The detailed design of a hardware implementation for *T has started this year. As an engineering decision we have chosen to place what amounts to two abstract nodes on each physical node. The proposed overall structure of a physical *T node is shown in Figure 1. As is shown in the figure, the node contains two 88110MP's, two memory controller chips (MC's), a network router, and DRAM.

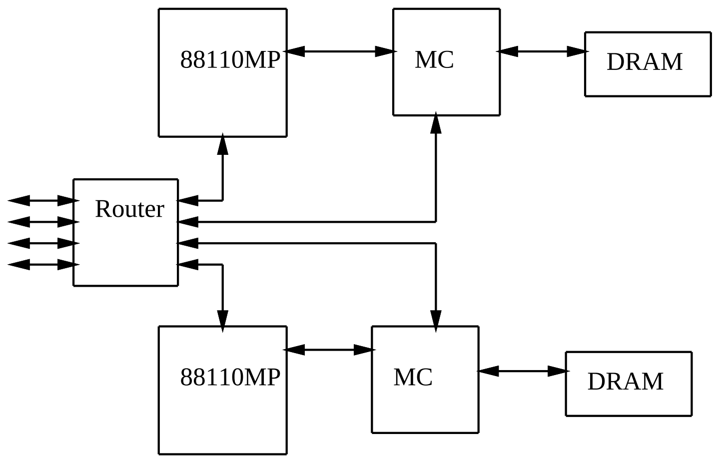

Figure 1: Implementation of a *T node

88110MP is the designation that we have given to our modified version of the Motorola 88110. The 88110MP contains all the functionality of both a Data Processor and of a Start Processor. It also contains a high performance network interface.

Each MC has the functionality of a RMem Processor. It also contains a high performance network interface.

The router is used to transfer packets among the components of the node and to transfer packets to the global interconnection network. The global network is constructed using the same type of router chip.

We have split the detailed design tasks with our industrial partner, Motorola. Motorola is responsible for the detailed design of the 88110MP while MIT is responsible for the design of the MC including caching strategies and the design of the network router.

9.1 Memory controller

Greg Papadopoulos, Jack Costanza, and Ralph Tiberio have started the design of the MC.

The MC integrates three primary node functions: control of local dynamic RAM, servicing of local 88110MSU reads and writes, servicing of remote load, store and DMA operations. The remote memory services are implemented by an on-chip microcoded protocol processor.

The MC also integrates a number of incidental functions, for example an interrupt controller and local bus arbiter. One 88110MSU's and an I/O controller are attached to an 88110 compatible 64-bit local bus. Up to four banks of dRAM are driven by the MC over a 128-bit datapath plus 16 bits of error correcting code. Finally, the MC is directly connected to two 16 bit wide high speed network ports.

With a 50Mhz node clock, the local bus, dRAM and network ports each provide a peak transfer rate of 400 megbytes per second. The MC incorporates smart on-chip cache and write buffer (the L2$) in attempt to sustain simultaneous full rate memory requests from the local bus and network interfaces.

The MC **local bus** is an 88110 compatible external bus interface comprising 64 bits of data, 32 bits of physical address and 48 control and information signals. The principal role of the MC is to act as the bus arbiter and to respond (as an *external device*) to physical memory read and write transactions sourced by bus masters. The MC is also a bus master as necessary to implement data cache coherence protocols.

The MC incorporates an interface to receive and transmit internode network packets. The interfaces are 16 bits wide, low voltage swing CMOS running at 2x the node clock. The network interface supports a modest amount of transmit and receive packet buffering. The network interface is internally connected to the **protocol processor**, an extensible microcoded engine capable of handling a variety of message types.

As dRAM controller, the MC has responsibility for address multiplexing, refresh timing, etc. The MC has control registers that allow the timing parameters associated with dRAM operation to be programmable. These registers allow the MC to be used with a wide variety of memory components. Also, the timing of the control signals generated by the MC can be tightened as dRAM technology improves, allowing for local memory upgrades. In addition to control, the MC also provides ECC to the local memory.

In terms of implementation status, we expect to complete a behavioral model of the MC, written in Verilog, sometime in August of 1992. After extensive simulation of the behavioral model, work will begin on the RTL model, which will ultimately be processed by synthesis tools to the gate level.

9.2 Caching strategies

Masato Motomura has studied caching strategies for *T.

Caching is one of the key technologies used for performance improvement in most modern computers. However, multithreading has a negative impact on cache hit rates because multithreaded architectures usually make frequent context switches. Our research goal is to find an appropriate cache strategy for multithreaded architectures and for *T parallel computer system.

The 88110MSU has separate 8KB data and instruction 1st level caches (L1$s), which are too small to maintain working sets of memory references. We are trying to augument these 1st level caches by a 2nd level cache (L2$) built in the memory controller (MC) chip. Since we don't have much area in the chip for this L2$, this cache should be intelligent enough to give high hit rates with small memory capacity.

The first design issue is how to exploit locality among threads. To this end, we are investigating ideas like stream buffering (prefetch), large block size (demand fetch), etc. The second issue is how to serve remote memory requests efficiently. We believe that the presence bit references associated with remote memory requests have plenty of locality which gives high hit rates for an L2$. Finally, maintaining memory coherency is an important task of the L2$ because remote memory request refer only to an L2$ not to an L1$. This means we should keep duplicated L1$ tags and copy back an L1$ line if this line has been modified and remotely requested. One interesting idea is to utilize these L1$ tags to maintain the exclusion property of cache hierarchy, i.e., excluding lines stored in an L1$ from an L2$. Given the fact that an L2$ will have same order of memory capacity as an L1$, this exclusion property might help increasing the hit rates of an L2$.

We have written a simulation environment for an L2$. Final design for an L2$ will be completed by the end of this summer after extensive simulation runs.

9.3 Router

Andy Boughton, Jimmy Kwon, Gowri Rao, and Satoshi Asari have started the design of the interconnection network for *T.

The global interconnection network for *T is a fat tree. The router on each *T node acts as the leaf stage of the tree.

Each router has eight sets of links. Each set is composed of a link in each direction. The router on a node uses four sets of links to communicate with the four major components of the node. It uses the other four sets to connect to the global interconnection network.

Each link is capable of transmitting 200Mbytes per second. The link uses a 16 bit wide data path.

A single reference clock is distributed throughout the *T system and the global interconnection network is operated in a synchronous manner. However each router contains circuitry that eliminates the need to carefully tune the electrical length of each transmission line. This circuitry allows each router input port to measure the phase of its incoming link at system setup time and to configure itself to safely receive data from the link.

Packets range in size from 128 bits to 768 bits. Each packet contains a 16 bit CRC. The CRC is checked by each router that the packet traverses.

The routing of each packet is predetermined at the source node. The route toward the root of the tree is randomly selected. The route back to the leaves is a function of the desired destination. A simple algorithm can be used to route around known bad links.

We expect to complete the behavioral specification of the router during the summer of 1992. We expect to have prototype router chips fabricated by the Fall of 1993.

10 Other work

The following work are closely related to our overall project developing parallel architectures and languages.

10.1 Network interface studies

Dana Henry and Chris Joerg have been studying network interfaces. Their proposed interface architecture typically achieves a three fold improvement over the best existing interfaces.[10] Most of the performance gain comes from simple, low cost hardware support mechanisms for fast dispatching on, forwarding of, and replying to messages. The remaining improvement is gained by mapping the network interface directly to the processor's register file rather than its memory. These mechanisms increase the performance of the interface without detracting from its flexibility. Using these hardware mechanisms, a register-mapped interface can receive, process, and reply to a remote read request in a total of just two RISC instructions.

This work grew out of the work on the Network Interface Chip (NIC)[9]. This chip, which has been designed and simulated at the RTL level, allows a Motorola 88100 processor to efficiently send and receive messages over the same communication network as used in Monsoon. Even though this chip will not be fabricated, it will have a considerable impact on the *T project. Motorola is incorporating many of the ideas from this chip and the related performance studies into a message co-processor which is being added to the 88110. This modified 88110 will be used as the basis of the *T processing nodes.

10.2 Synchronization studies

Shuichi Sakai studied synchronization and pipelining in massively parallel computer systems. Dr. Sakai analyzed currently existing data-driven synchronization mechanisms from the viewpoint of efficiency and hardware complexity. He proposed an optimized synchronization mechanism and a pipeline structure for a massively parallel computer using this mechanism. He also developed performance improvement methods for this pipeline. Dr. Sakai's work is described in [13].

10.3 Parallel alpha-beta search

Bradley C. Kuszmaul has been working on developing a highly-parallel alpha-beta search algorithm. The algorithm exploits dataflow techniques, such as non-strict function evaluation, to obtain parallelism. At this point, he has achieved good parallel speedup for alpha-beta search. However, the algorithm currently uses too much memory to be practical; Bradley is working with Prof. Charles E. Leiserson on reducing the memory requirements of the algorithm. Bradley is also engaged in joint work with Prof. Hans Berliner of CMU to apply this highly parallel alpha-beta search algorithm to a real chess program.

Ahmed Shah worked as a UROP on chess, with Bradley C. Kuszmaul. In Summer '91, Ahmed implemented a naive chess program in ID. The program exhibited some parallelism, and played legal chess (but very badly) using the TTDA simulator. During January '92, Ahmed implemented an improved user interface for a chess program written in C.

Publications

Theses Completed

S.B.

S.M.

Ph.D.

Theses in Progress

Brobst, Stephen A. "Storage Management in a Tagged Token Dataflow Machine," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected February 1993.

Chery, Yonald "Dataflow Graph Partitioning and Threading for the Monsoon Processor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected September 1992.

Iyengar, Arun "Dynamic Storage Allocation on a Multiprocessor," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected September 1992.

Kuszmaul, Bradley C. "Compiling Data-Flow Programs for Control-Flow Computers" Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected December 1992.

Sharma, Madhu "Design and Evaluation of a Multi-thread Processor Architecture," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected December 1992.

Lectures

References

[1] Z. Ariola and Arvind. Compilation of Id. In *Proc. of the Fourth Workshop on Languages and Compilers for Parallel Computing, Santa Clara, California, Springer-Verlag Lecture Notes in Computer Science 589*, August 1991. (Also: CSG Memo 341).

[2] Z. Ariola and Arvind. Graph rewriting systems. In *Proc. Symposium on Semantics and Pragmatics of Generalized Graph Rewriting, University of Nijmegen, The Netherlands*, December 1991. (Also: CSG Memo 323).

[3] Z. M. Ariola. *An Algebraic Approach to the Compilation and Operational Semantics of Functional Languages with I-structures.* PhD thesis. Ph.D. thesis, Harvard University, June 1992.

[4] H. Barendregt, M. van Eekelen, J. Glauert, J. Kennaway, M. Plasmeijer, and M. Sleep. Term Graph Rewriting. In *Proceedings of the Parallel Architectures and Languages Europe Conference, Eindhoven, The Netherlands, Springer-Verlag Lecture Notes in Computer Science 259*, pages 141–158, June 1987.

[5] A. P. W. Böhm and R. E. Hiromoto. The Dataflow Complexity of Fast Fourier Transforms. submitted to the Dataflow workshop of the 19th International Symposium on Computer Architecture conference.

[6] A. P. W. Böhm and R. E. Hiromoto. Developing Dataflow Algorithms. In *Proceedings of the 13th World Congress on Computation and Applied Mathematics, Dublin*, 1991.

[7] W. Farmer and R. Watro. Redex Capturing in Term Graph Rewriting. Technical Report M89-59, MITRE corporation, Massachusetts, 1989.

[8] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, New Jersey, 1971.

[9] D. Henry and C. Joerg. The Network Interface Chip. Technical Report CSG Memo 331, MIT Laboratory for Computer Science, Cambridge, MA, June 1991.

[10] D. Henry and C. Joerg. A Tightly Coupled Processor Network Interface. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992.

[11] J. Kennaway, J. Klop, M. Sleep, and F. de Vries. Transfinite Reductions in Orthogonal Term Rewriting Systems. In *Proc. RTA '91, Springer-Verlag Lecture Notes in Computer Science 488*, 1991.

[12] J.-J. Lévy. *Réductions Correctes et Optimales dans le Lambda-Calcul.* Ph.D. thesis, Université Paris VII, October 1978.

[13] S. Sakai. Synchronization and Pipeline Design for a Multithreaded Massively Parallel Computer. Technical Report CSG Memo 343-1, MIT Laboratory for Computer Science, Cambridge, MA, March 1992.

[14] J. P. Singh, W. Weber, and A. Gupta. SPLASH: Stanford parallel applications for Shared-Memory. Technical report, Computer Systems Laboratory, Stanford University, april 1991.

[15] C. Wadsworth. *Semantics And Pragmatics Of The Lambda-Calculus.* Ph.D. thesis, University of Oxford, Semtember 1971.

[16] P. Welch. Continuous Semantics and Inside-out Reductions. In *λ-Calculus and Computer Schience Theory, Italy (Springer-Verlag Lecture Notes in Computer Science 37)*, March 1975.

Contents

1 Introduction **2**

2 Personnel and Visitors **3**

3 MIT-Motorola collaboration on Id, Monsoon, and *T **4**

4 Other external collaborations **5**

4.1 DEC Cambridge Research Lab 5

4.2 Berkeley 5

4.3 Sandia 5

4.4 Los Alamos 5

4.5 Colorado State University 6

5 Id: general topics **7**

5.1 Graph Rewriting Systems: capturing sharing of computation in language implementations 7

5.2 Partial Evaluation 9

5.3 Basic Input/Output 9

5.4 High level Input/Output library 10

6 Id: compiler and run-time systems for Monsoon **11**

6.1 Storage reclamation in Id 11

6.2 Efficient Implementation of Loops 13

6.3 Efficient manipulation of literal values 14

6.4 Frame manager 15

6.5 Heap manager 15

6.6 Id World 16

6.7 Id compiler in Id 16

6.8 Test suites 19

7 Monsoon hardware **19**

8 Applications and performance measurements **19**

8.1 Matrix Multiply, Gamteb, Simple and Paraffins 19

8.2 B-trees 20

8.3 Water 20

9 *T hardware **21**

9.1 Memory controller 22

9.2 Caching strategies 23

9.3 Router 24

10 Other work **24**

10.1 Network interface studies 25

10.2 Synchronization studies 25

10.3 Parallel alpha-beta search 25