LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Computation Structures Group Progress Report

July 1, 1989 *to* June 30, 1990

Computation Structures Group Memo 309
June 1, 1990

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, 1989 — June 30, 1990

Academic Staff

Arvind *(Group Leader)*
J.B. Dennis
R.S. Nikhil
G.M. Papadopoulos
A. Vezza

Research Staff

G.A. Boughton
J. Young
R.P. Johnson

Graduate Students

S. Aditya
P.S. Barth
S.A. Brobst
Y. Chery
D. Chiou
S. Glim
D.S. Henry
J.E. Hicks
A.K. Iyengar
C.F. Joerg
V.K. Kathail
B.C. Kuszmaul
J.S. Onanian
A. Salaam
M. Sharma
A. Shaw
K.M. Steele

Undergraduate Students

G. Adams
A. Caro
J. Ferrera
S. Furman
R. Lustberg
F. Madero
S. Parekh
D.B. Plass
D. Stetson
R. Tewari
C. Tse

Technical Staff

J.P. Costanza
R.F. Tiberio

Support Staff

S.M. Hardy
F. Ugwuegbu
B. Radle

Visitors and Adjunct Members

A. Altman (Texas Instruments)
Z. Ariola (Harvard University)
M. Heytens (Dept of EECS, MIT)
R. Kreuter (Siemens, Munich)

Computation Structures Group

1 Introduction

Our group is interested in general-purpose parallel computation. Our approach is centered on

- declarative, implicitly parallel languages.
- dataflow architectures, which are scalable because of their tolerance of increased memory latencies and support for frequent synchronization. Our vehicles for research include an abstract "Explicit Token Store" architecture (ETS), hardware prototype implementations of ETS (called Monsoon), various software emulators (GITA, MINT), and a software emulator for a new proposed architecture called P-RISC.
- sophisticated compiling and run-time systems for Id, both for dataflow and other architectures. We have also explored the use of dataflow compiling for an experimental persistent programming language.
- applications programs to guide the language, compiler and architecture research.

Last year, we reported that we had begun negotiations with Motorola for a project to produce, as a research prototype, a complete system running Id on dataflow machines using the Monsoon processor architecture. This year, MIT-Motorola cooperation has moved into high gear. This involves extensive and daily cooperation in the design and production of the Monsoon hardware, and in the design and production of the Id programming environment. Two-node prototypes are expected by the end of summer, 1990, and 16-node machines by spring, 1991. To this end, a formal cooperation agreement has been signed, and Motorola has established and staffed a new "Motorola Cambridge Research Center" at One Kendall Square, next door to LCS.

Our main research vehicle for programming languages is Id, which has fine-grained, implicit parallelism. We have been able to formalize our incremental typing system for Id and to prove it correct. We have made much progress in developing the "manager" construct in Id, which is a disciplined way of using imperatives while retaining fine-grained, implicit parallelism and synchronization. We have continued our work in formalizing Id's operational semantics in terms of abstract reduction systems. New applications in Id include the Travelling Salesman Problem using simulated annealing, the Viterbi Search from speech recognition systems, and various sparse matrix algorithms.

We have made almost a complete transition from the TTDA (Tagged Token Dataflow Architecture) compilation schemas to new schemas that incorporate the notion of frame storage in an integral way. Frame storage is now used for extensive loop optimizations. A new back-end translates these frame-oriented dataflow graphs into code for Monsoon. We have been studying resource management for Id in great detail, including compiler-directed garbage

collection, as well as numerous versions of frame and heap managers for improved concurrency.

The porting of Id World to the Unix environment (from our original Lisp machine environment) is complete, and has been distributed outside MIT.

The Monsoon wire-wrap prototype, which has now been running for over a year, has been invaluable for testing out our ideas in resource management in Id, for measuring instruction mixes and for designing its successor.

The second generation Monsoon processor has been designed using various ASICs. Motorola has done the board-level design, and is fabricating it. The processor incorporates substantial improvements from the wire-wrap prototype in speed, functionality and connection to the Unix world. An I-structure board has also been designed and is being fabricated by Motorola. The Monsoon interconnection network switching chips (PaRCs) and data link chips (DLCs) have been designed and fabricated, and are undergoing testing. A 4-by-4 network board has been designed by Motorola and is being fabricated.

In other work: Kathail has completed his Ph.D. thesis on optimal interpreters for the lambda-calculus; we have continued our research on P-RISC, a synthesis of von Neumann and dataflow architectures; we are close to having a stock hardware implementation of Id; and, we are close to having a dataflow implementation of a parallel persistent language.

In addition to the cooperation with Motorola, we continue to maintain strong and active contacts with several other dataflow researchers outside MIT. Members of our group participated in the international committee that designed the new, standard, functional programming language Haskell.

2 Personnel and Visitors

In January 1990, Greg Papadopoulos was appointed to the MIT faculty in the Department of Electrical Engineering and Computer Science. He has been a member of our research staff since August 1988,

In December, Arthur Altman of Texas Instruments completed a year as a visitor in our group, and is now a visitor in Prof. Steve Ward's group.

Rudiger Kreuter from Siemens, Germany, spent the Fall of 1990 as a visitor in our group. In addition to learning about Id and dataflow, he studied the implementation of 3D graphics in Id.

As usual, we have had a steady stream of international scholars for short visits and talks.

3 MIT-Motorola collaboration on Id and Monsoon

Through the concerted efforts of Al Vezza, Associate Director of LCS, and Jerzy Skibinski, Vice President of Motorola's Microcomputer Division, a joint Research Agreement with

Motorola's Computer Division of Tempe, Arizona was formalized in August 1989, although cooperation had been ongoing for seven months in anticipation of the signing.

The joint effort will result in at least three 16-node Monsoon research prototypes and at least sixteen 2-node versions. The division of labor between MIT and Motorola is as follows: MIT is responsible for the overall system, logic and chip designs, chip fabrication, a novel special tool for generating microcode from opcode specifications, the Id language and compiler design and development. Motorola is responsible for all board level, enclosure, supporting hardware infrastructure and I-structure logic design, development and manufacturing. On the software side, Motorola is responsible for the Monasm assembler, dynamic linking loader, command line interpreter user interface, all host level software, and debugging tools including a Monsoon simulator.

Motorola's project is managed by Jim Richie. Their hardware work is done at their facility in Arizona, while their software work is done mostly in Cambridge at the new Motorola Cambridge Research Center (MCRC), which they have established as part of this project. The immediate focus of MCRC, which is in the Kendall Square office complex next to LCS, is close cooperation with MIT in the research and development of software for Monsoon. In the long run, MCRC is expected to take its place alongside the many fine basic research labs in the vicinity of MIT. The first employee of MCRC was Ken Traub, who completed his Ph.D. in our group in 1988. Ken was the original architect and builder of our Id compiler. As a member of MCRC, Ken is playing a leading role as overall architect of the Monsoon software system.

During the year, we have held numerous review and planning meetings with Motorola:

- August 1, 1989: Review meeting, at MIT.
- September 27-28, 1989: Software and Contracts meeting, at MIT
- October 19, 1989: Review meeting, at MIT.
- December 7, 1989: Monsoon technical discussion meeting, at MIT.
- January 25-27, 1990: Monsoon hardware and software progress review meeting, at Motorola, Tempe, AZ.
- March 29-30, 1990: Review meeting, at MIT.
- June 25, 1990: Monsoon hardware and software progress review meeting, at Motorola, Tempe, AZ.

We are happy to report that all critical milestones to date have been met. We expect that the first 2-node prototype will be available in 3rd Q/90 and the first 16-node prototype in 2nd Q/91.

4 Other external collaborations

Our work on Id and Monsoon has led to collaborative efforts with many research groups outside MIT.

On leaving MIT after finishing his Ph.D., Bob Iannucci has started the Empire project at IBM Research, whose goal is to build a hybrid dataflow-von Neumann machine similar to the one he proposed and studied here in his thesis. We also continue to collaborate with K. Ekanadham of IBM Research, who is leading the effort to target our Id compiler for that machine. In the summer of 1989, Shail Aditya worked at IBM Research on their Id compiler.

At Sandia, a group of researchers led by Gerald Grafe is building the Epsilon dataflow machine which is similar to Monsoon in many respects. Jamey Hoch of Sandia is working on retargeting our Id compiler for Epsilon. In addition, they will be using our PaRC network switching chip in the interconnection network for their multiprocessor. Ken Steele has just left our MIT to join the group at Sandia. We have participated in several meetings to discuss collaboration with the Sandia project:

- July 31, 1989: Sandia-DARPA meeting, at Washington D.C.
- March 11-13, 1990: Cooperation meeting, at Sandia, Albuquerque, NM.

Karl Ottenstein at Los Alamos and Bob Ballance at University of New Mexico are working on a compiler for FORTRAN on Monsoon. They plan to use the back-end of our Id compiler. Similarly, Keshav Pingali at Cornell is also investigating the implementation of imperative languages on a dataflow machine— he, too, plans to use the back end of our Id compiler in order to run his codes on Monsoon.

In a separate, but related activity, Arvind, Rishiyur Nikhil and Jonathan Young were members of the design team for Haskell, the new, non-strict functional programming language. The team included about 15 prominent researchers in functional programming from the U.S. and Europe. The report on Haskell was published in April, 1990. It is hoped that the international research community will adopt this language as the standard for non-strict functional languages.

On November 1-3, 1989, we held a Dataflow Workshop here at MIT. In addition to researchers from all the above groups, participants included recent graduates from our group, and researchers from Yale, Manchester University, Tera Computers, Rice University, Oregon Graduate Institute, Glasgow University, Motorola and Hewlett Packard Labs.

On April 26-27, we held a Software Cooperation Meeting here at MIT, again attended by researchers from most of the above groups. The focus was on discussing how each of us could structure our work to maximize sharing, since many of us are interested in targeting other languages to Monsoon and in targeting Id to other machines.

One of the outcomes of the Software Cooperation Meeting was a consensus among our guests that we needed to run a workshop dedicated to furthering the understanding of the internal structure of the Id compiler. This workshop has been scheduled for, June 28-29, 1990 at MIT.

On February 2 1990, we held an internal (MIT) workshop on multi-threaded architectures with participants from our group and the groups led by Prof. Anant Agarwal, Prof. Steve Ward, Prof. Bill Dally, Prof. Tom Knight, as well as Bert Halstead from DEC Cambridge Research Lab. The intent was to get a better understanding of each others' work, since we are all exploring different kinds of multi-threaded architectures.

As usual, on July 24-28, 1989, the summer dataflow course (6.83s), was taught here at MIT by Professors Arvind and Nikhil. The course was attended by approximately 20 external researchers.

In November, 1989, Professors Arvind and Nikhil taught a one-day tutorial on Id at the Supercomputing '89 Conference in Reno, Nevada.

5 Id: general topics

5.1 Types and incremental type-checking

Continuing his work on the incremental type inference system for Id, Shail Aditya developed an abstract model for incremental property maintenance and applied it to show the correctness of the incremental type inference system developed for Id.

Incremental programming environments, such as LISP, aim at providing the user the flexibility to write a sequence of definitions constituting the program, one by one and in arbitrary order, resolving global references to other definitions dynamically. They allow editing and testing parts of an incomplete program or debug those parts that are incorrect, without worrying about the status of the rest of the program. However, the Hindley/Milner static type inference system [6, 2] followed in Id does not naturally lend itself to incremental compilation. Nikhil in [7] discussed the issues involved and outlined a high-level mechanism to do it.

Following Nikhil's proposal, Shail Aditya devised an abstract scheme to adapt the Hindley/Milner type inference system for incremental compilation. Subtle incremental interactions were discovered between the types of a given set of definitions and their partitioning into strongly connected components (SCC), definitions that are mutually recursive with each other. Development of the necessary theoretical framework guided modifications in the scheme to handle polymorphic and mutually recursive definitions correctly. Essentially, the present scheme consists of maintaining an upper and a lower type bound for each toplevel identifier along with its current SCC. Inconsistencies arising due to the declared type falling out of the expected range or because of a change in its SCC are detected and the affected definitions are flagged for recompilation. The goal is to show an exact correspondence between the types inferred in the incremental scheme with those inferred when a complete and correct program is given while at the same time performing minimal recompilation work due to an incremental change in the program. The detailed proofs of the correspondence are due to appear in Shail Aditya's forthcoming master's thesis. Future work in this direction will be to optimize the space and time requirements of the incremental book-keeping done by the compiler.

5.2 Managers

Paul Barth and Rishiyur Nikhil continued their work on managers. Managers add nondeterminism to Id, an important property for state-sensitive computation, as required by operating systems, databases, and I/O. Although nondeterminism is a powerful feature, it introduces a new class of programming errors: irreproducible results. We are addressing this at the language level by encapsulating managers in abstract type definitions. A manager is an abstract type, consisting of an updatable state and a set of operations that access and update the state. Each operation computes a new state from the old state. Mutual exclusion is provided for the state so concurrent operations do not interfere. This encapsulation allows state invariants to be proved by proving them for each operation.

Several challenging efficiency concerns are now being addressed. For space efficiency, managers with a complex state should mutate the state rather than copy it. For parallelism, such managers should provide fine-grained mutual exclusion, so independent operations can proceed concurrently. These concerns raise syntactic and semantic issues in the design and implementation of managers. A new syntax has been proposed that addresses these issues while maintaining a clean abstraction.

Manager applications have been written for graph algorithms, sorting, memory management, union-find, and parallel priority queues. The traveling salesman problem was coded using a simulated annealing algorithm, using managers for both path mutation and random number generation. A detailed study of potential parallelism and synchronization bottlenecks was performed on several variations of the algorithm.

5.3 Sequentialized Code Execution

We are currently experimenting with writing resource managers and operating systems in Id. These kinds of programs, which make use of imperative side-effects, must have explicit sequentialization of reads and writes.

James Hicks has extended the Id language and compiler for sequential constructs. The new syntax sequentializes the execution of groups of bindings in let blocks or loops. To sequentialize a group of bindings, use '&' instead of ';' to separate the bindings, as shown in this example:

```
{ x0 = e0;
  x1 = e1
& x2 = e2;
  x3 = e3
in
  e4 }
```

This ensures that the evaluation of e2 and e3 does not begin until all computation has ceased in the previous two bindings. Note, however, that e0 and e1 may execute in parallel, and

that e2 and e3 may execute in parallel — sequentialization is only inserted between binding groups separated by '&'.

Parentheses may be used to group bindings and enforce more arbitrary synchronization graphs. Here is an example:

```
{ x0 = e0;
  ( x1 = e1 &
    x2 = e2 )
  in e3 }
```

In this example, e0 may execute in parallel with e1 and e2, but e2 may not begin execution until expression e1 terminates.

5.4 Formalization of Id's operational semantics

Zena Ariola and Arvind have continued their work on giving a precise operational semantics for Id. The approach consists of translating Id into a simpler and smaller kernel language, and giving semantics of the kernel language in terms of an Abstract Reduction System (ARS). In order to prove the correctness of compiler optimizations, a notion of program equality is needed. Such a notion is easier to define for an ARS than an interpreter. P-TAC, an earlier attempt to define such a language and ARS was reported last year [1].

P-TAC was a simple and a low-level language that allowed us to capture most aspects of the current implementation of Id on the dataflow machines. However, it was so far from the source language that the translation procedure (from Id to P-TAC) became a serious impediment in understanding the operational semantics of Id. Even though it allowed many program optimizations to be described in terms of source-to-source P-TAC program transformations, it ruled out certain other program optimizations because the information to perform them was essentially lost in the translation process.

Kid, our current kernel language, is essentially a de-sugared version of Id-, which is Id without comprehensions, general union types and pattern matching, and non-deterministic features such as managers. Both array and list comprehensions can be expressed in terms of other Id features such as loops and "open" lists. Though such a translation is not simple, it can be understood in its own right. Similar remarks apply to complex pattern maching. The stage at which type-checking should be performed is still a open question. Given the type definitions, type checking can be done at the Kid level though it may be profitable to do so at an earlier stage.

An ARS for Kid, which includes nested function definitions and loops, has been defined. Many loop optimizations and partial evaluation have been expressed as Kid source-to-source transformations. The work on formalizing the printable output and termination of Id programs is under progress.

6 Id: compiler and runtime systems for Monsoon

6.1 New compilation schemas for dealing with frames

James Hicks implemented the code generator for the bounded loop schema for the Monsoon back end. The TTDA bounded loop schema is much different from the Monsoon bounded loop schema because each iteration executes in a different context on Monsoon, while in TTDA only the iteration number within a context changed — this necessitates a change in the D operator that routes tokens from one iteration to the context, or activation frame, of the next iteration. Another change is that the synchronization that allows only k iterations to execute at once must be performed using locks and two-phase transactions — this synchronization was performed with bit-vectors and special instructions in Gita.

The new bounded loop schema consists of three parts: setup, iteration, and cleanup. The setup portion consists of a 1-bounded, or sequential, loop that allocates a ring of activation frames and fills in the loop constants in each frame. The iteration portion consists of the actual loop body plus the glue necessary for synchronization and to route tokens to the next iteration or to the outputs of the loop. The cleanup portion of the loop clears and deallocates each iteration context. We have taken much care so that the setup, iteration and cleanup portions of the loop may be overlapped, to reduce the latency incurred by loops that execute few iterations. The setup portion allows the iterations to begin as soon as it has one activation frame setup. When the loop predicate evaluates to false, the cleanup portion of the code is triggered with the continuation of the next context in the ring, which is guaranteed to be inactive at that point. The cleanup code starts with that context, and continues around the ring, with the proper synchronization to ensure that it does not overrun the loop body.

6.2 Staging the Instruction Set Development for Monsoon

The macroinstruction set of Monsoon is a "soft" interface, such that an opcode is successively decoded through the pipeline by downloadable lookup tables. The decode tables are set up by a host processor whenever Monsoon is cold-booted. In Monsoon, an opcode encodes the effective address mode, the matching mode (*e.g.*, join, constant, imperative), the ALU operation and the number and disposition of result tokens. For example, the double-precision floating point subtract operation `FSUB` consumes 32 opcodes for all of its variants of one *vs.* two outputs, constant *vs.* dyadic matching, *etc.*

The software Monsoon interpreter, `MINT`, is also indirectly driven by the decode tables. A pre-processing program, `MUD`, takes the decode tables as input and, for each opcode, produces a C (originally Lisp) subroutine which is later compiled and linked into `MINT`.

In order to manage the "bring-up" and validation of the compiler, the software which generates the decode tables, and `MINT`, we have partitioned the instruction set into three subsets, to be developed in stages. The first set, *IS0*, has approximately 60 opcodes (out of a possible 2048) and represents a very minimal instruction set. *IS0* supports frame and I-structure al-

location, but no deallocation and only single deferred readers. Exceptions are not supported, nor is the runtime type system (Id is statically typed).

The next stage, *IS1*, brings the total to a few hundred opcodes and is capable of supporting the entire Id lanaguage, including closures, accumulators, managers, storage reclamation, and multiple deferred reads against I-structures.

The final stage, *IS2*, encompasses the whole instruction set, including experimental extensions for temporary registers and threading. As of June 1990, the *IS0* set has been certified by a series of tests on our gate level simulator of a Monsoon processing element, and an *IS0* version of the compiler and `MINT` has executed a simple successive over-relaxtion 2D wavefront problem.

6.3 New back end for Monsoon

This past year Andrew Shaw has implemented a new back end for the ID compiler to transform ID program graphs into Monsoon machine language. Prior to this, we had been generating code for the Monsoon wire-wrap prototype using an interim back end that was a modification of the original TTDA (Tagged Token Dataflow Architecture) back end. The new back end uses the same data-structures as the middle end of the compiler, and several new optimizations have been implemented, along with the standard peephole optimizations that were in place with the old TTDA back end. For example, the calling convention constrains the entry-points of procedures to lie in consecutive address locations; one of the new modules can relax the conservative selection of these instructions. Since the Monsoon architecture has some assembly constraints on the layout of machine code, some new algorithms were designed to enforce these constraints upon the final output code. For example, two-output instructions are constrained to have one of their destination instructions in the following instruction slot; a new bipartite-matching algorithm was implemented to find a near-optimum selection of successor-constrained instructions.

In addition, an interim loader was implemented to interface with the new Monsoon Interpreter, since the full loader has not yet been implemented.

6.4 Compiler-directed storage reclamation for Id

We have been experimenting with structure-storage management over the past year. Informal studies have shown that functional languages typically "cons" at 4 times the rate of Lisp programs. This high rate of storage allocation means that functional programs have a great dependency on garbage collection. Unfortunately, garbage collection can be very expensive, especially on a parallel machine.

We have introduced a pragma, `@Release`, into Id to annotate structures that are temporary and that should be deallocated. When the Id compiler sees an `@Release` annotation on a structure, it inserts code to deallocate the structure upon termination of the nearest enclosing conditional branch, procedure body, or loop iteration.

There is one further optimization the compiler can perform with **@Releases**. If a structure is allocated in a loop, and deallocated in the same or next iteration, then the compiler can lift the allocate and deallocate out of the loop to reduce the overhead of calls to the storage manager. Outside the loop, k copies of the structure will be allocated, where k is the loop bound. These will be used by the iterations of the loop. After the loop terminates, all k structures will be deallocated.

The **@Release** pragma has been used extensively by Olaf Lubeck in his Id implementations of the Gamteb photon transport benchmark and the Particle-in-Cell (PIC) code. In October, 1989, Lubeck, Hicks and Johnson got Gamteb to run on the Monsoon prototype. This version of Gamteb was annotated so that it did not leak any storage — all structures that became garbage were deallocated. The largest problem that was run on the prototype started with 40,000 particles. It allocated 300,000 9-tuples, 200,000 3-tuples, and 270,000 activation frames of size 512. When it completed only 616 words of storage we still allocated — and that contained the answer. This is quite impressive considering that the prototype only has 128K words of memory, and only half of that is used for the heap. This work has shown that explicit structure-storage management is useful — it allows us to run programs that could not be run otherwise.

James Hicks is working on compiler analysis for the verification and automatic insertion of **@Releases** as his PhD research. The goal of this work is to have the compiler analyze programs to determine the lifetime of structures, and to insert code to deallocate structures that are no longer needed. The compiler performs lifetime analysis by using abstract interpretation— it interprets the program over an abstracted value domain at compile time, in order to determine which expressions in the program allocate structures, and where those structures may be used.

In scientific codes, which tend to have very regular control and data flow, most, if not all, of the structures that become garbage should be detected and deallocated by the compiler. Hicks has performed some experiments with simple program analyzers that support this belief. His analyzer detected 23 of the 25 **@Releases** inserted into Gamteb by Lubeck. The analyzer also was run on Simple; in this case the compiler inserted deallocates that at run time deallocated 85% of the garbage created by 4 iterations of Simple on a 10 by 10 grid. By the end of fall 1990, the compiler should be able to insert code to deallocate all of the garbage created by this program.

6.5 Run time systems for Id

Jonathan Young has led a major effort this year on developing the Id Run Time System (RTS). The RTS is designed to be a flexible interface to the low-level primitives needed to execute Id programs. The RTS is composed of three main parts: the allocation and deallocation of *contexts*, both fixed- and variable-size, for procedure invocations, the allocation and deallocation of *aggregates* for data structures, and the management of input from and output to the outside world. Most of our work this year has focused on the first two parts.

As a stopgap measure until we have a simulator capable of executing SVC trap instructions, we have coded primitive heap and context allocators for the new Monsoon machine which

maintain two pointers to the beginning and end of the heap. Since they do not reuse storage, when the pointers cross, the machine dies. These allocators are generated inline by the compiler, and have allowed the rest of the software development to proceed.

We expect that as IS1 (instruction set level 1, including SVC instructions) becomes operational on the simulator, we will be able to test and debug the full functionality of the two-phase operations and the exception mechanism. Once this happens, the Id RTS can begin to execute via SVC instructions, although most of the RTS handlers will simply make a procedure call at this point. When registers are finally added to the simulator (IS2), we expect that the RTS will become much more efficient.

There are several problems to be addressed in order to manage storage efficiently on a Monsoon multiprocessor. In particular, we must avoid network traffic – most of the manager code must be completely *local*. We also wish to ensure that the critical sections are short and the reuse of memory is as high as possible.

On the new Monsoon machine, each processor will allocate contexts locally, and, using the exception mechanism, each thread must be able to allocate a context independent of the other threads in the pipeline. We have designed and implemented (but not debugged) a scheme which, on average, achieves this behavior by cacheing a small number (16) of contexts with each thread while linking the rest into a processor-global context free-list.

Under this scheme, each allocation (and deallocation) of fixed-sized contexts will take approximately 6 instructions (exception, load cache pointer, return fetched context to caller, increment pointer, store pointer) normally and 20 instructions for the exceptional case that the free (or empty) list has over- or underflowed. Since this happens statically once every 16 operations, the amortized cost averages out to no more than 7 instructions.

On the new architecture we aim to solve several new problems when allocating objects from the global heap. First, the heap will be *interleaved* across multiple nodes in the system. While no work is needed to achieve this interleaving, the heap manager will need to create pointers which take full advantage of the hardware interleaving mechanism.

Second, multiple processors will be handling multiple simultaneous heap requests. Even though the heap is remote, we desire to handle the majority of all requests locally. This requires some PE-local heap data structures. Finally, we wish to avoid interference between different threads executing in parallel on the same PE.

The heap manager on the Monsoon multiprocessor is a hybrid of two schemes. Each processor will run a *local allocator*, a version of quick-fit which utilizes the assumption that if an object is deallocated, it is likely that another object of the same size will be allocated. When the local allocator runs out of memory, however, it will ask the *global allocator* for more storage, pre-allocating a large block of memory for future requests. While consing off the global heap does require network traffic, this should be tolerable because it is so infrequent.

Arun Iyengar has also begun to be actively involved in designing the Id storage managers.

7 Applications

Paul Barth and Stephen Brobst implemented a number of approaches to parallel simulated annealing for the traveling salesman problem. Manager extensions to the ID programming language were used to facilitate the implementation of critical sections while performing update-in-place operations on the adjacency matrix of cities in the algorithm. A number of paradigms for the use of managers in implementing the algorithm were explored: Compare and Swap, Canonical Ordering, Master Lock, and Locking with Back-Off. It was found that fine-grained parallelism was exposed very naturally in the model of execution provided by dataflow. However, it was found that there is significant coarse-grain sensitivity within the program to the particular algorithm implemented for managing critical sections. It was important to not over-serialize execution of loops corresponding to different temperatures in the simulated annealing algorithm. However, it was also critical to consider the contention resulting from too many parallel loop iterations. This contention resulted from two sources. One source was the contention for the current seed value of the random number generator. This problem was addressed by initiating multiple, parallel random number generators. The other source of contention was on the cities to be swapped. In general, to swap two cities, it is required to grab locks on six cities (the two to be swapped as well as the two neighbors for each city). The methods of managing this locking had a large impact on performance. As expected, any use of global locks introduced a major synchronization bottleneck for large instances of the problem. By optimizing the "fast path" of execution through the locking primitives we were able to reduce, but by no means eliminate, the impact upon the critical path of our computation.

Amin Salaam and Rishiyur Nikhil have been studying an implementation of the Viterbi Search in Id. Viterbi Search is a key component of speech-recognition systems. The inputs to the search are an Accoustic-Phonetic network (APnet) and a dictionary. The APNet is a graph generated by an earlier phase that performs signal processing on the accoustic signal of the speech utterance. Each arc in the graph represents a time interval in the utterance, and is labelled by a list of probabilities, indicating how well the signal in that time interval matched each of all possible phonemes. The dictionary is also a graph, structured around words. Each word is represented by a sequence of arcs corresponding to phonemes, and words that can legally appear in sequence are also connected by arcs. Dictionary arcs are also labelled with probabilities indicating possible omission of insertion in an actual utterance. The Viterbi Search algorithm matches paths in the APnet to paths in the dictionary, finally emitting the "most likely" sequence of words in the utterance. Because of this complex graph structure, the algorithm has not been effectively parallelized to date. We have been studying existing code (in C and Scheme) for this algorithm since March 1990 in order to extract the fundamental aspects of the algorithm. We plan to produce a parallel implementation in Id and to run it on Monsoon. We expect that this will shed much light on dataflow implementations of graph algorithms in general.

Other applications written by Rishiyur Nikhil and Arvind in Id include: a simulator for a very abstract model of an Explicit Token Store (ETS) dataflow machine (of which Monsoon is a concrete example), and various versions of LU-decomposition for dense and sparse matrices.

8 Id World, the Id programming environment

Development of Id World, the Id programming and experimentation environment continues with a focus on advances which benefit both the old TTDA/GITA system and the software system which will support Monsoon. Paul Johnson has worked on improvements in Id World on Unix workstations and software system construction tools. Id Mode for Gnu Emacs provides source code indentation and compilation of Id programs. With the assistance of Hicks, simulator statistics graphs and overlays of multiple graphs are available under the X Window System. Id World version 4.3, which was released in April, provides these improvements and Id compiler pragmas for explicit structure storage management. As of version 4.1 Id World runs on Unix workstations under common lisp. Version 4.3 has been tested under Allegro Common Lisp and Lucid Common Lisp running on Mips, Motorola, and Sun workstations. We were unsuccessful in running Id World under Austin-Kyoto Common Lisp (AKCL) due to deficiencies in language support for error handling. Improvements in software system construction tools include: handling filesystem specifics – translation of program filenames and logical pathname support in the Defprogram facility, startup initializations and banner customization for Lucid and Allegro disk images, consistent versions of internal software – generation and use of generic patch files for our development systems.

9 Monsoon Hardware Development

9.1 Monsoon Wire-wrap prototype processing element

The Id compiler now produces Monsoon object code for all of the Id language. The only restriction to running an Id program on the prototype is the size of memory — the prototype has only 128K words of data memory, of which half is used for I-structure storage and half for activation frame memory.

We have a run-time system, written in Id and Monsoon assembly code by Young and Hicks, that manages the allocation and deallocation of activation frames and heap storage on the Monsoon processor. Activation frames are managed on a free list, and heap storage is managed by either the buddy or first-fit system. Implementation of the storage managers has been very difficult because concurrent calls to the storage manager can be interleaved on an instruction-by-instruction basis.

The prototype is currently used to measure performance and instruction mixes of applications under the Monsoon instruction set architecture with the run time system described above. These measurements have shown that Id can be run efficiently on Monsoon. The dynamic instruction mixes collected while running Id applications on Monsoon have shown that the architecture is well matched to the language, and not too much overhead is introduced to support instruction level parallelism. However, some of the architectural restrictions imposed by the implementation of the wire-wrap processor, such as the restriction of one explicit destination per instruction and 10-bit offsets for addressing of frame locations, have caused excessive difficulty in compilation and have introduced excessive overhead in the object code. The processor has been redesigned to mitigate these problems.

These performance measurements have also shown that better algorithms must be used for storage management so that concurrent invocations of the storage manager will not cause excessive sequentialization of the program. This sequentialization will be magnified on multiple-processor Monsoon systems unless better algorithms are used. Young and Armando Fox have designed and implemented a storage manager that uses multiple local free lists and local heaps to allow concurrent access to the heap.

9.2 The Monsoon processing element (2nd generation)

We have made significant progress in the design and verification of a second-generation Monsoon processing element (PE). This board-level design was transfered to the Motorola Microcomputer Division, where it was sucessfully layed-out and routed. It is now is the process of being manufactured. We also designed and fabricated two application-specific integrated circuits (ASICs): a byte-slice of the datapath and tag/pointer ALU. Gregory Papadopoulos was responsible for the overall PE and ASIC architectures. Jack Costanza and Ralph Tiberio executed the detailed designs and simulations.

The original Monsoon wire-wrap prototype processor was made operational in 1988, executing its first compiled Id program in October of that year. The new PE is largely compatible with the original prototype, employing an eight-stage pipeline, 64-bit datapaths (plus 8 bits of type), and 32-bit instructions. The new PE differs in several important respects, however:

- **Interprocessor Network.** The first prototype could only operate as a uniprocessor because it lacked an interprocessor network. The new PE integrates the network into the processor by employing an on-board PaRC and Datalink chips, as well as input and output FIFO buffers.

- **VME Interface.** The new PE is hosted on a 9U form-factor VME bus. The VME bus interface implements diagnostic functions (access to processor scan state, single stepping, breakpointing), VME and PE interrupts, and high speed I/O through a dual ported frame store.

- **Exception Handlers.** The new PE now provides a way to very efficiently transfer control to state-preserving exception handlers. Exceptions can be induced unconditionally (an SVC), elicited by a standard ALU conditions (like overflow) or by operand type inconsistencies. In fact, we will use the SVC mechanism to provide dynamic linking to the resource management system, including frame and heap allocation/deallocation.

- **Temporary Registers.** The new PE permits state to be communicated between an instruction and its successor through a small set of temporary registers associated with the first stage of the ALU. There are eights sets of temporaries with three 72-bit registers per set, one set for each "logical thread" being interleaved by the eight stage pipeline. It is expected that temporaries will measurably improve the dynamic efficiency of compiled code.

Experience generating code for the wire-wrap prototype has suggested a number of minor enhancements to the processor datapath. For example, it is now possible to use the current tag as one of the arguments to the ALU— optimizing procedure linkage and **case** statements. We have also attempted to make the design more manufacturable by providing complete scan coverage of all internal processor state and parity protection for instruction memory, frame store and the token queues. Figure 1 gives the block diagram of the processing element datapath. The processor is designed to run on a 100 nanosecond cycle time, yielding a sustained processing rate of 10 million tokens per second. Compiled code presently exhibits a dynamic average of 1.4 tokens per instruction. Thus, the processor pipe should deliver approximately 8 million instructions per second, any fraction of which may be floating point operations. The first set of processors will be equipped with 256Kwords (32-bits) of instruction memory, 256Kwords (72-bits) of frame store memory and 64Kwords (144-bits) of token queue memory. Both the instruction and frame store memories are upgradable to 1MWord.

The processing element detailed design was captured and extensively simulated on our Apollo/Mentor Graphics design systems. Both arrays were implemented in LSI Logic's 10K series of 1.5 micron channeless arrays, and packaged in 144 pin fine-pitch quad flat packs. The DATAPATH array comprises a little over 10,000 gates and implements a 9-bit slice of the datapath pipeline registers, temporaries, breakpoint registers, form token multiplexers, VME interface, and all static RAM parity generation and check. Each processor uses eight DATAPATH options. The PIU gate array is an ALU function unit specialized for tag and pointer manipulation. The PIU array comprises approximately 6,500 gates. Both arrays have been successfully prototyped in volumes sufficient for an initial build of five processing elements.

Design verification emphasized full-board gate-level simulation and timing verification, including all of the gate arrays. Simulation tests generally took the form of small hand-coded dataflow graphs designed to test various aspects of the instruction processing mechanism. One or more initial tokens would be introduced into the pipeline, and then the simulation was allowed to "free-run" using the processing of the tokens themselves to generate the various test vectors. The simulated design was transferred to Motorola in the Fall of 1989. A 12-layer surface-mount circuit board was sucessfully routed by Motorola in February of 1990, and assembly began on the first processor prototypes in May 1990. Simulation and verification of the processor still remains an area of intense activity. We have generated a set of code fragments with reference timing traces to aid in the hardware debugging and certification process.

9.3 The interconnection network for Monsoon

Andy Boughton, Christopher Joerg, Juan Ferrera, and Robert Lustberg have continued development of the network for Monsoon. This network is packet switched and supports a bandwidth of 800 MBits/sec/port. The two primary components of the network are the Packet-switched Routing Chip (PaRC) and the Data Link Chip (DLC). These components were discussed in some detail in last year's progress report.

PaRC is a CMOS gate array designed by Chris Joerg that forms the basis of the Monsoon

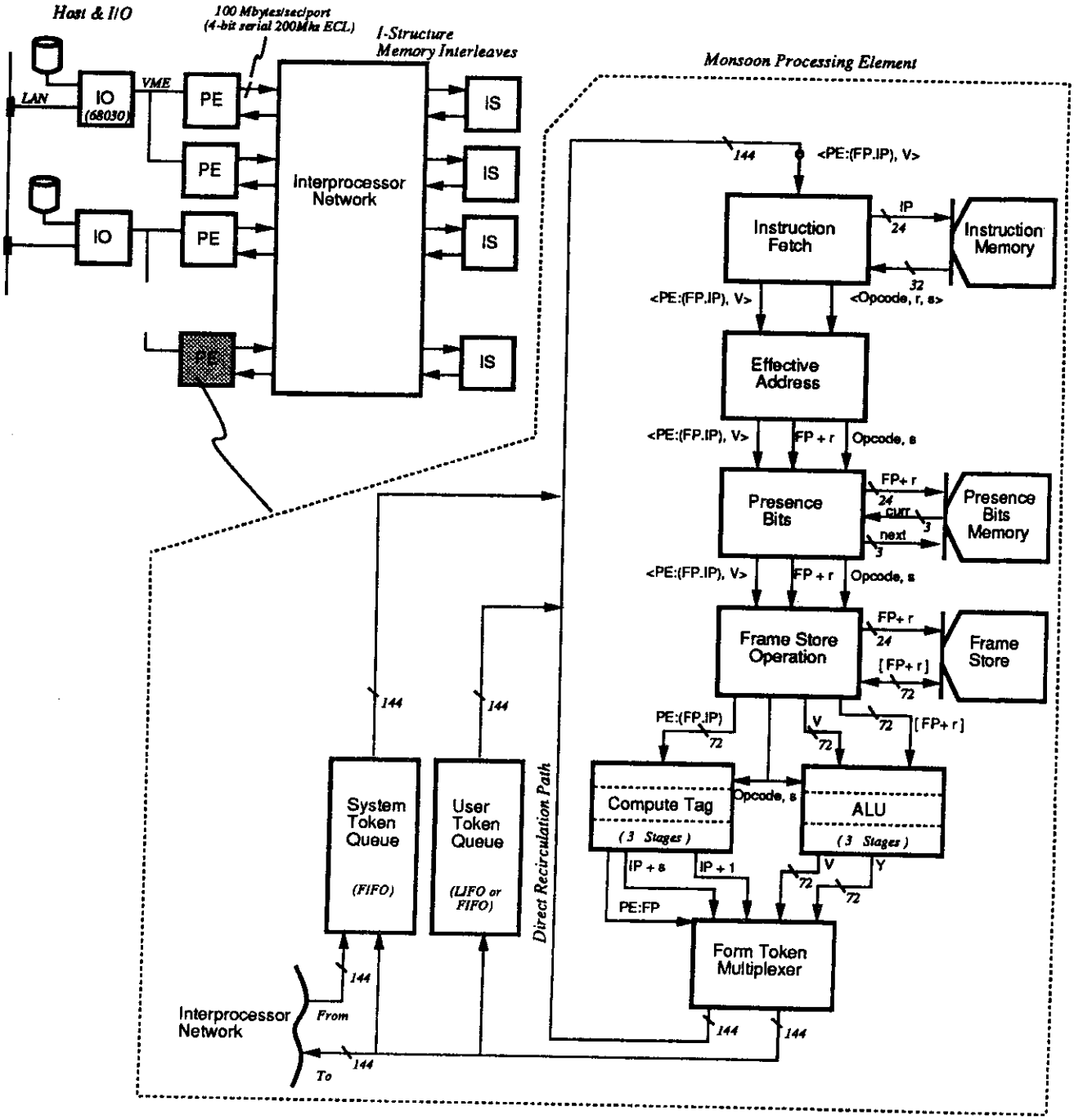

Figure 1: Block diagram of Monsoon processing element datapath

network. PaRC has 4 input ports and 4 output ports, each of which is 16 bits wide and has a maximum throughput of 800 MBits per second. Each input port has 4 buffers, each of which can hold one packet. PaRC has a sophisticated buffering and scheduling strategy which will allow an output port to transmit a packet whenever possible. PaRC uses a CRC code to detect errors on received packets. PaRC also allows a processor to get a fast acknowledgment that its message has been received. The mechanism for this is able to provide the acknowledgment without further burdening the network.

Joerg has finished the design of PaRC and the generation of a complete set of test vectors. PaRC has 33,000 used gates and is capable of operating at 50 MHz. It has a low latency (100ns in light traffic), while making effective use of its bandwidth (90% utilization in heavy traffic). The set of test vectors allow the vendor to check for defects on newly fabricated chips.

PaRC has been fabricated in LSI Logic's 1.5 micron compacted array series and working chips have been received. One of these chips has been placed on a simple test fixture constructed by Juan Ferrera. This setup was used to verify the timing of selected output signals.

The DLC is an ECL gate array that interfaces 16 bit wide PaRC ports to 4 bit wide interboard cables. Each DLC contains one data link transmitter and one data link receiver. Each of the 4 bits of the interboard data path is differentially driven at 200 MBit/sec.

Andy Boughton has completed the design of DLC and the generation of a set of test vectors. DLC has been fabricated in Motorola's Mosaic II ECL array series and working chips have been received.

Juan Ferrera, and Robert Lustberg have designed and constructed a test fixture for the DLC chip. The fixture uses two DLCs to transmit test patterns over a 40' data link cable. Tests with this fixture indicate that DLCs can be used to reliably interconnect network boards in different racks.

The first use of PaRC and DLC will be in the initial version of the Monsoon processor board. These boards are currently under construction. Each of these boards uses one PaRC chip and one DLC chip. These boards will allow PaRC and DLC to be more thoroughly tested in an operational environment.

4 input 4 output network boards using a PaRC and 4 DLCs have been designed by our industrial partner, Motorola. These boards should be available early in 1991. These boards will be used in the construction of 16 node Monsoon systems.

9.4 The I-structure memory board

Ken Steele completed his master's thesis titled "Implementation of an I-Structure Memory Controller" in February 1990. The design was kept simple to reduce cost and design time. In particular, the board does not perform local management of deferred continuation lists. Instead, the compiler allocates storage *in the frame* for one cell of a deferred list. When an I-fetch instruction is executed against an empty I-structure location, the I-structure board automatically responds with a token whose continuation is a small modification of the normal

return continuation. The effect of this modified continuation is to thread the deferred list through all the frames from which the deferred I-fetches were issued.

Motorola Microcomputer Division in Tempe, Arizona is building an I-structure controller based on Steele's thesis design. Fabrication of the hardware is currently underway.

9.5 Caching for Monsoon

This past year Derek Chiou has been investigating the possibility of caching on Monsoon. The prototype Monsoon uses static RAM for all of its memory at the present time. It would be desirable to use slower, cheaper dynamic RAM for future iterations of the processor. If the processor is to run at competitive speeds, using slower RAMs will require caching of some sort. Software has been developed that will produce an instruction trace from either MINT or the Monsoon prototype. Derek also wrote a cache simulator which takes an instruction trace and collects cache data. Most of the serious data collection has been done for very small caches – generally around 32 words of fully associative cache. Results have been reasonably promising, with hit rates ranging from 33% to 84%. These results are very preliminary, however.

9.6 Completion of MINT, a Monsoon simulator

Last year we reported the construction of MINT (Monsoon Interpreter), a simulator of the Monsoon instruction set. MINT has a variety of uses: debugging microcode for Monsoon, development and testing of Monsoon object code generators in the compiler, gathering of more detailed statistics than Monsoon itself is capable of gathering, *etc.*

This year, Andrew Shaw has extended MINT to simulate multiple Monsoon processor execution. The extension has a network simulation that models latency, but not contention. It is expected that contention will not be a significant problem as the performance of the Monsoon network is very high, and the references will be well distributed, as data is interleaved across processors. A multiple-processor run-time system was implemented to distribute processes and to handle resource-manager requests. Several experiments were run that indicate that GITA simulation was indeed an accurate predictor of performance in Monsoon. In addition, the capacity of the network was deemed sufficient to handle processors' memory requesets, and that the limiting factor in parallel execution is likely to be serialization induced by requests to the resource-manager. In addition to multiple-processor simulation, an I-structure board simulation was added to MINT.

10 Other activities

10.1 Optimal Interpreters for the Lambda-Calculus

Vinod Kathail completed his doctoral dissertation [4] in which he developed a new interpreter for the λ-calculus that is *optimal* in the theoretical sense defined by J.-J. Lévy [5] and gave

proofs of its correctness and optimality.

The interpreter is based on a new graph representation for λ-expressions that permits sharing of not only subexpressions but also *contexts*, *i.e.*, parts of an expression that are not complete subexpressions. This is in contrast to the commonly used representations of expressions, which permit sharing of only subexpressions.

The interpreter is presented as a graph reduction system along with a normalizing strategy for applying the reduction rules. The set of rules includes a graph version of the β-rule of the λ-calculus as well as certain other rules, some of which are similar to the rules for handling environments in an environment-based interpreter for the λ-calculus. Some of the nice features of the interpreter are as follows. First, all the reduction rules are local constant-time operations on graphs. Second, the reduction strategy for applying the rules is quite simple. Finally, the input to the interpreter as well as the output of the interpreter are "clean" representations of λ-terms; they don't contain various new types of nodes used by the interpreter. A version of the interpreter has been implemented on lisp machines.

To prove the correctness of the interpreter, the thesis develops two calculii, called λ_{fc} calculus and λ_f calculus. λ_{fc} calculus is essentially the term version of the graph reduction system underlying the interpreter. λ_f calculus is obtained from λ_{fc} calculus by removing certain types of terms and reduction rules that are not very useful for terms. The thesis shows the correspondence between the graph reduction system underlying the interpreter and λ_{fc} calculus as well as correspondence between the two calculii and De Bruijn notation [3]. Although λ_f calculus was motivated by the interpreter, it may be of general interest because of the way it simulates changing of De Bruijn numbers.

The thesis also strengthens an earlier result of Barendregt *et al* that states that if λ-expressions are represented as trees, then there is no *recursive* (one-step) reduction strategy that is optimal. The extension proved in the thesis provides some justification for the basic assumption underlying the optimality criterion, *i.e.*, the number of β-contractions performed in reducing an expression is a good measure of the cost of reducing the expression.

10.2 P-RISC

Madhu Sharma is investigating the design of a processor that is a concrete implementation of the P-PRISC architecture. Most multi-threaded architectures incur a large context-switching cost. The cost may be incurred either in *hardware*— when register space is provided for a large number of contexts, or in *time*— when a contexts have to be swapped in and out of processor register files. The proposed design virtually eliminates both elements of context-switching cost. It caches contexts in multiple register set in the processor, but manages to mask the context swapping cost using an additional port to the register file and deep, deterministic, instruction lookahead mechanism. The design is claimed to be only marginally more expensive than commercial RISC processors such as the SPARC.

A detailed simulator for the architecture has been developed. Statistics gathered for small hand-coded programs run on the simulator indicate that the architecture does manage to mask context-switching cost and performs well under low or high parallelism.

We are now developing compiling techniques for the architecture. Two approaches will be pursued. The first is a *dataflow-graph* driven approach, wherein we start with a dataflow graph, sequentialize threads of the computation to obtain larger threads (whenever there is no gain in executing the threads in parallel), and arrive at a "control-flow graph", which is translated into P-RISC code. The second approach is the conventional "control-based" approach and will be used for compiling imperative languages.

10.3 Compiling Id for von Neumann machines

Bradley Kuszmaul has been working on retargeting the Id compiler for stock hardware, such as conventional Unix workstations. Starting with the existing dataflow graphs produced by the compiler, he translates these into parallel control flow graphs based on the P-RISC abstract machine model. The major effort is then in analysis and transformations on the control flow graph, including strictness analysis, subscript analysis, identification of threads, transformations to lengthen threads and reduce synchronizations, and peephole optimizations. Finally, these graphs are used to generate object code in the T language (T is a dialect of Scheme). The existing T compiler already has a very sophisticated code generator for a variety of stock machines (including register allocation, closure optimization, *etc.*).

We expect to release a version of Id World using this new compiler by June 30, 1990. This implementation should substantially increase the availability of Id World to researchers who may not have access to Lisp machines or the Monsoon dataflow machines. It should shed light on the differences in implementation requirements between non-strict, lenient languages like Id, and non-strict, lazy languages like Miranda.

10.4 Parallel persistent languages

Michael Heytens and Rishiyur Nikhil have made significant progress in their project to design and implement a parallel persistent language. The aim is to produce a system in which (a) the the user can declare, create and manipulate objects of arbitrary structured types and (b) all such objects are automatically persistent. Such a system can be viewed as a synthesis of programming languages and databases.

Because of the rich object structure of the language, and because the structure can change significantly over time, high performance cannot be achieved using conventional database methods (detailed planning of data layouts on disks and scheduling of disk activity). Our approach is to use parallelism.

We have designed a kernel database language that is greatly inspired by Id, *i.e.*, having fine-grained, implicit parallelism. In addition, in update transactions, each field can only be redefined once, as in I-structures; this allows update transactions also to be run in parallel. We have implemented a compiler that translates transactions into dataflow graphs and then into P-RISC code that is augmented with manager calls for disk I/O. As in Id and Monsoon, another objective is to mask disk latencies using parallelism.

We have designed a segmented, paged, distributed virtual heap for the persistent system. Pages of the virtual heap reside in files in each processing element, and are fetched on demand into page frames in each processing element; the protocols for page faults and flushing on transaction-commit have been designed. The filespace occupied depends only on the heapspace in use (not the entire virtual heap address space). The files are partitioned across processing elements, and are partitioned by type, allowing fast traversals of collections of objects of a given type. The files may also be indexed, allowing fast direct access to individual objects.

A prototype is being implemented, consisting of an ensemble of P-RISC emulators running on a network of Sun workstations. It is currently running a subset of the language, and we expect to support the full language by June 30, 1990. After this, we plan extensive evaluations, running numerous published and new benchmarks, and porting the system to a real multicomputer with parallel disks.

10.5 Bachelor thesis and UROP projects

Armando Fox (supervised by Jonathan Young) investigated possibilities for more efficient runtime storage managers. A number of traditional schemes were analyzed, with particular attention to projected performance and synchronization requirements for a dataflow architecture. A prototype of a storage manager was implemented which addressed the problems of global resource contention and long critical sections, and its performance was compared to the existing first-fit manager. Although substantial improvements were observed in a variety of cases, overhead associated with clearing out storage for reuse still accounted for a significant fraction of the deallocation latency. Increasing the efficiency of this operation and possibly implementing some sort of garbage collector remain topics for future investigation.

David Plass (supervised by Jonathan Young) implemented a parser generator which produces LALR shift-reduce tables in the dataflow language Id, for use in an Id parser. The algorithm employed avoids the creation of the LR(1) kernels by calculating LR(0) kernels and later adds LALR lookahead information. In addition, a general-purpose parser shell was implemented which can be used in conjunction with output by a compiler to parse source language inputs. This work will help in our effort to write the Id compiler in Id.

Alejandro Caro (supervised by Jonathan Young) designed and implemented a symbolic debugger for the Id programming language on the Monsoon processor. The debugger allows the user to trace function calls and returns, to examine local variable bindings and loop variable bindings, and to examine the state of the machine in detail. Furthermore, the debugger allows the user to invoke these functions at the *source code level*, relieving the user from having to learn the intricacies of the processor and compilation schemes.

Glen Adams, (supervised by Greg Papadopoulos), extended the LISP MINT simulator to model I-structure memory. An I-structure module simulator is capable of handling I-store, I-fetch, I-put, and I-take requests, as well as ordinary reads and writes. This was interfaced to the MINT system so that it is suitably initialized and driven by the queuing system. The design is extensible to model multiple I-structure units.

Mike Flaster (supervised by Rishiyur Nikhil) implemented a compiler for a small, non-strict language that uses dependence analysis to convert a non-strict program into a strict one. First, the compiler performs conventional def-use analysis, as well as subscript analysis for arrays in loops, using the Banerjee-Wolfe and GCD tests, augmented with some symbolic subscript-expression reduction. Then, the compiler performs loop-splitting, loop-reversal, loop-distribution, scalar expansion, induction-variable analysis, *etc.* to ensure that all dependencies are forward-dependencies. The program can now be run with the parallelism that is best suited to the resouces of a given machine.

UROP student Doug Stetson (supervised by Jonathan Young) implemented a compiler which translates a small subset of C into an Id Program Graph, the intermediate language of the Id Compiler. The subset included assignments, conditionals and loops, but not procedure calls or pointers. The sequential semantics of C was enforced by artificial dataflow edges.

Publications

Arvind and Nikhil, R.S. "A Dataflow Approach to General-purpose Parallel Computing," Computation Structures Group Memo 302, MIT Laboratory for Computer Science, Cambridge, MA, July 1989. Prepared for the proceedings commemorating the 25th Anniversary of Project MAC.

Arvind and Nikhil, R.S. and Pingali, K. "I-Structures: Data Structures for Parallel Computing," *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 4, October 1989. Also: Compuation Structures Group Memo 269, March 1989.

Arvind and Nikhil, R.S. "Executing a Program on the MIT Tagged-Token Dataflow Architecture," *IEEE Transactions on Computers*, Vol. 39, No. 3, March 1990. Also: Computation Structures Group Memo 271, June 1988.

Hudak, P. and Wadler, P. (editors) "Report on the Programing Language Haskell, A Non-strict Purely Functional Language (Version 1.0)," Yale University, Department of Computer Science - YALEU/DCS/RR777, April 1990. Authors include Arvind and Rishiyur Nikhil and Jonathan Young.

Iyengar, A.K. "Parallel Characteristics of Sequence Alignment Algorithms," In *Proceedings of Supercomputing '89*, Reno, Nevada, November 13-17, 1989, pp. 304-313. Also: Computation Structures Group Memo 304, October 1989.

Kuszmaul, B.C. and Fried, J. NAP (No ALU Processor): The Great Communicator, *Journal of Parallel and Distributed Computing*, Vol. 8, No. 2, February 1990, pp. 169 - 179.

Nikhil, R.S. "New Languages with Implicit Parallelism and Heap Storage are Needed for Parallel Programming," In *Opportunities and Constraints of Parallel Computing*, Jorge L.C. Sanz (ed.), Springer-Verlag 1989, ppgs. 93-95. Position paper presented at workshop, December 5-6, 1988, IBM Almaden.

Nikhil, R.S. and Arvind "Id: a language with implicit parallelism," Computation Structures Group Memo 305, MIT Laboratory for Computer Science, Cambridge, MA, February 1990.

Nikhil, R.S. and Arvind "An Abstract Description of a Monsoon-like ETS interpreter," Computation Structures Group Memo 308, MIT Laboratory for Computer Science, Cambridge, MA, April 1990.

Papadopoulos, G.M. "Program Development and Performance Monitoring on the Monsoon Dataflow Multiprocessor," In *Instrumentation for Future Parallel Computing Systems*, Frontier Series, ppgs. 91-110. Also: Computation Structures Group Memo 303, October 1989.

Papadopoulos, G.M. and Culler, D.E. "Monsoon: An Explicit Token Store Architecture," In *Proceedings of the 17th International Symposium on Computer Architecture*, Seattle, Washington, May 1990. Also: Computation Structures Group Memo 306, March 1990.

Young, J. "Tests for Monsoon Instruction Subset 0," Computation Structures Group Memo 307, MIT Laboratory for Computer Science, Cambridge, MA, April 1990.

Theses Completed

S.B.

Adams, Glen. "Adding I-Structure Simulation to Mint," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

Caro, Alejandro. "An ID Debugger," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

Flaster, Michael "The Use of Dependence Analysis to Improve Compilation of Non-strict Languages" S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

Fox, Armando. "Parallel Storage Allocation in ID," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

Plass, David B. "PIDGEN: a Parser Generator in ID," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

S.M.

Joerg, Christopher. "Design and Implementation of a Packet Switched Routing Chip" S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

Steele, Kenneth M. "Implementation of an I-Structure Memory Controller," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. January 1990.

Wang, George L. "A Synchronization Network used for Debugging and Diagnostics on the Monsoon Multiprocessor System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. June 1990.

Ph.D.

Kathail, Vinod K. "Optimal Evaluators for Lambda-calculus Based Functional Languages," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. May 1990.

Theses in Progress

Aditya, Shail "An Incremental Type Inference System for the Programming Language Id," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected August 1990.

Henry, Dana S. "Analysis of Real-time Constraints in Stream Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected August 1990.

Hicks Jr., James E. "Compiler directed storage reclamation by object lifetime analysis," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected May 1991.

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 1990.

Lectures

Arvind. "P-TAC: A Parallel Intermediate Language," IFIP Conference on Functional Programming Languages and Computer Architecture, Imperial College, London, United Kingdom, September 12, 1989.

Arvind. "Project Dataflow," Presentation at MIT to DARPA visitors, October 26, 1989 and December 5, 1989.

Arvind. "Dataflow Architectures and Languages," Distinguished Lecturers Series, University of Southern California, November 17, 1989; Indian Institute of Science, Bangalore, India, December 15, 1989; Indian Institute of Technology, Madras, India, December 18, 1989; Indian Statistical Institute, Calcutta, India, December 19, 1989.

Arvind. "Supercomputing, Parallelism and Dataflow," Supercomputing '89 Conference, Teraflop Computer Panel, Reno, Nevada, November 14, 1989; Luncheon for Professor Emeriti at MIT, November 9, 1989.

Arvind. "Multi-Threaded Architectures: An Internal Workshop," Presentation at MIT, Laboratory for Computer Science, February 2, 1990.

Arvind. "Declarative Programming and Implicit Parallelism," Parallel Computing Workshop, Ohio State University, Columbus, Ohio, March 21, 1990.

Arvind. "Architecture Challenges to Realizing Massive Parallelism," Parallel Computing Workshop, Ohio State University, Columbus, Ohio, March 23, 1990.

Arvind. "Explicitly Programmed Parallelism versus Automatically Generated Parallelism," Panel discussion at Parallel Computing Workshop, Ohio State University, Columbus, Ohio, March 22, 1990.

Arvind. "Does Programming Parallel Machines Have To Be Painful?," Presentation at MIT-EECS Colloquium Series, April 9, 1990.

Arvind. "Advances in Dataflow Architectures: The Monsoon Project," Activities in Computing and Artificial Intelligence, Workshop at MIT for NEC Corporation and Laboratory for Computer Science, May 7, 1990; School of Computer Science, Systems Seminar Series, McGill University, Montreal, Canada, April 20, 1990; IBM, Austin, Texas, June 5, 1990; Lawrence Livermore Laboratory Sisal Workshop at Asilomar State Park, Pacific Grove, CA, June 7, 1990; IBM Research, Almaden, CA, June 8, 1990.

Arvind. "Computer Architecture: What It Can Do and What It Cannot Do for Artificial Intelligence," Conference on Knowledge Based Computer Systems: KBCS '89, Bombay, India, December 11-13, 1989.

Arvind. "Measuring Fine Grain Parallelism," Workshop on Advanced Programming Environments for Scientific Computation, Yale University, New Haven, CT, June 11, 1990.

Arvind. "The Evolution of Dataflow Architectures from TTDA to P-RISC," Massive Parallelism: Hardware, Programming and Applications, Amalfi, Italy, October 9, 1989.

Brobst, S.A. "Software Pipelining and VLIW Architectures," Boston University, Boston, MA, December 6, 1989.

Kathail, V. K. "An Optimal Interpreter for the λ-calculus," MIT Dataflow Workshop, MIT, Cambridge MA, November 1989; Bellaire Research Center, Shell Development Company, Houston, TX, March 1990; University of Chicago, Chicago, IL, April 1990; Cornell University, Ithaca, NY, May 1990.

Kathail, V. K. "A λ-calculus Interpreter that Implements Lévy's Optimal Reductions," Semantics of Graph Reduction Workshop, Ecole Normale Supérieure, Paris, France, April 1990; INRIA, Rocquencourt, France, April 1990.

Iyengar, A.K. "Parallel Characteristics of Sequence Alignment Algorithms, Supercomputing '89, Reno, Nevada, November 13-17 1989.

Nikhil, R.S. "Compiling for P-RISC," 2nd Workshop on Languages and Compilers for Parallel Computing, University of Illinois, August 1989.

Nikhil, R.S. "The parallel programming language Id and its compilation for parallel machines," Workshop on Massive Parallelism, Amalfi, October 10, 1989.

Nikhil, R.S. "Compiling the parallel language Id for P-RISC, a dataflow RISC machine," McGill University, Montreal, October 27, 1989.

Nikhil, R.S. "Progress with P-RISC," Dataflow Workshop, MIT, November 3, 1989.

Nikhil, R.S. "LU decomposition in Id," Lecture # 3, Tutorial on "Programming in Id," Supercomputing '89, Reno, Nevada, November 13, 1989.

Nikhil, R.S. "The Parallel Language ID and its compilation for P-RISC, a dataflow architecture," University of Pennsylvania, Philadelphia, December 1989. Boeing Computer Services (Helicoptor support), Philadelphia, December 1989. NECUSE Workshop on Parallel Computing, Amherst College, Amherst, January 8, 1990. (NECUSE - Northeast Consortium on Undergraduate Science Education)

Nikhil, R.S. "Another way to compile a non-strict language," Meeting of the IFIPS Working Group 2.8 on Functional Languages, Rome, Italy, March 26-29, 1990.

Nikhil, R.S. "Id Update," Monsoon Software Cooperation Meeting MIT, April 26, 1990.

Papadopoulos, G.M. "Scan Path Design," Motorola Microcomputer Division, Tempe, Arizona, October 24, 1989.

Papadopoulos, G.M. "Dispelling the Dataflow Myth: There is still a program counter...Lots of program counters!," MIT Seminar, October 30 1989.

Papadopoulos, G.M. "Monsoon as a Multithreaded Multiprocessor," MIT Dataflow Workshop, November 2, 1989.

Papadopoulos, G.M. "The Olympic Games Information System," Olympic Games Candidacy Committee, Athens, Greece, March 9, 1990.

Papadopoulos, G.M. "Monsoon: An Explicit Token Store Architecture," International Symposium on Computer Architecture, Seattle, Washington, May 29, 1990.

Young, J. "Compiling ID for a Real Dataflow Machine," Compass, Cambridge, MA, February 23, 1990.

References

[1] Z. M. Ariola and Arvind. P-tac: A parallel intermediate language. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, September 1989. Also: CSG Memo 295, MIT Laboratory for Computer Science 545 Technology Square, Cambridge, MA 02139, USA.

[2] L. Damas and R. Milner. Principal Type-schemes for Functional Programs. In *Proceedings of the 9th Symposium on Principles of Programming Languages*, pages 207–212, January 1982.

[3] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Proceedings of Koninklijke Nederlandse Akademie van Wetenschappen, Series A, Mathematical Sciences*, 75:381–392, 1972.

[4] V. K. Kathail. *Optimal Interpreters for Lambda-calculus Based Functional Languages.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1990.

[5] J.-J. Lévy. Optimal reductions in the Lambda-calculus. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 160–191. Academic Press, London, 1980.

[6] R. Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[7] R. S. Nikhil. Practical Polymorphism. In *Lecture notes in Computer Science*, volume 201. Springer Verlag, September 1985.

Contents