LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Computation Structures Group Progress Report 1990-91

Computation Structures Group Memo 337
June 6, 1991

G.A. Boughton (ed.)

This report describes research done at the Laboratory for Computer Science of the Massachusetts Institute of Technology. Funding for the Laboratory is provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-89-J-1988.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Computation Structures Group

July 1, 1990 — June 30, 1991

Academic Staff

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

Research Staff

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

Graduate Students

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

Undergraduate Students

S. Asari	R. Durham	R. Lustberg
L. Bader	A. D'Silva	S. Parekh
A. Chang	J. Ferrera	E. G. Perez
D. Choi	S. Kho	C. Tse
S. Chunawala	J. Kulik	R. Virk

Technical Staff

J. P. Costanza R. F. Tiberio

Support Staff

S. M. Hardy B. Radle

Visitors and Adjunct Members

Z. Ariola	(Harvard University)
K. D. Chung	(Pu San National University, Korea)
D. Hwang	(Sung Kyun Kwan University, Korea)
R. Narayan	(Indian Institute of Science, Bangalore)
S. Sakai	(Electrotechnical Laboratory, Japan)
T. Senta	(NEC, Japan)
M. Srinivasan	(Indian Institute of Science, Bangalore)

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 architectures.
- compilers and run-time systems for Id, targeting dataflow and other architectures.
- applications programs to guide compiler, language, and architecture research.

Last year, we reported on our collaborative project with Motorola to develop the Monsoon dataflow machine. This year, the project has borne fruit in the form of Monsoon hardware which is currently being used to support software development.

Substantial progress has also been made in the support software for the Monsoon system. The following components have been completed: a back-end for the Id compiler that produces instruction set level 2 (IS2) Monsoon code, the MINT interpreter capable of executing IS2 Monsoon code, the Monsoon loader, the MMI server which provides a uniform software interface to the Monsoon hardware and MINT interpreter, the MONASM assembler for Monsoon code, an execution manager, a statistics viewer for analyzing the statistics gathered during the execution of Monsoon code, and the Id World environment which integrates the compiler, the loader, the execution manager, the MINT interpreter, and the Monsoon hardware.

We continue to diversify our knowledge and experience in the field of implicitly parallel programming languages and their efficient compilation through our research language Id. Id has now matured into a more complete and more expressive programming language since last year, with the addition of "M-structures" for fine-grain non-deterministic computations, a parallel input/output facility, and a simple and efficient solution to overloading of operators. Our concerted efforts towards formalization of the language and the compiler semantics have exposed areas of potential performance improvements in the Id compiler and have given us insights into the compiler-directed storage reclamation issues. To test our understanding of the language semantics and its expressive power, as well as to create a faster, alternate vehicle for Id compilation, we have recently started two projects to write the entire Id compiler in Id.

With the availability of Monsoon hardware and its software simulator MINT, the Id language has advanced to become a system software language for Monsoon. The Id run-time system, consisting of a parallel frame manager, a heap (dynamic storage) manager, and several system-call handlers, is almost entirely written in Id and runs on the actual Monsoon hardware. Several conventional and novel storage management schemes are currently under investigation. We have conducted extensive tests of the Id compiler instruction set and the Id libraries on the hardware. We have also been able to run several large-sized, numeric and symbolic applications written in Id in conjunction with the run-time system, and have exciting results to report.

Our aim of providing a fine-grain, implicitly parallel, programming environment extends well beyond Monsoon. We are retargeting the current Id compiler for conventional von Neumann machines which involves extensive control flow analysis and innovative multi-threaded resource management and scheduling in order to achieve high performance. Independently, we are well on our way towards a new, standalone, portable Id compiler that will run on ordinary workstations. This uses the P-RISC abstract machine as its intermediate target that nicely captures the requirements of a parallel language on sequential architectures. Both the above projects have sharpened our understanding of compilation issues for high-performance, scalable, MIMD architectures, and will ultimately be used in our next generation architecture project, *T.

2 Personnel and Visitors

Christine Flood joined the group in January of 1991 and is working on Monsoon test suites and performance analysis.

Yuli Zhou joined the group in December 1990. Zhou is currently involved with the Id compiler in Id project. His research interests include compile time analysis of functional programs.

Kidong Chung has been visiting the group since March 1991 and will be with us until September 1991. Chung is studying dataflow languages and architectures and resource management schemes.

Daejoon Hwang has been visiting the group since July 1990 and will be with us until July 1991. Hwang is studying dataflow architectures.

Ranjani Narayan visited the group from September 1990 to March 1991 researching instruction clustering in the Id compiler.

Shuichi Sakai has been visiting the group since April 1991 and will be with us until April 1992. Sakai is playing a major role in the development of the architecture for the *T synchronization processor.

Tetsuhide Senta has been visiting the group since September 1990 and will be with us until August 1991 researching memory management schemes for the *T processor.

Mandyam Srinivasan has been visiting the group since April 1991 and he will be with us until October 1991 studying both the evaluation of parallel architectures and hardware design methodologies.

3 MIT-Motorola collaboration on Id and Monsoon

Our joint effort with Motorola is progressing very well. We are happy to report that all major milestones have been met. A Monsoon Processing Element (PE) was delivered by Motorola Tempe to Motorola Cambridge Research Center (MCRC) during the first calendar quarter of 1990 and one to MIT during the third calendar quarter. During the fourth calendar quarter an I-structure (IS) memory was delivered to MIT allowing us to upgrade the single PE system to a 2-node (1-PE, 1-IS) system. During the second calendar quarter of 1991, Motorola delivered two additional 2-node systems and a 16-node system consisting of 8 PE's, 8 IS's, an interconnection butterfly packet network, and appropriate enclosure with cooling and power supplies.

Now that the basic hardware systems are in place, both we and MCRC have been able to intensify our efforts to develop software for Monsoon. Motorola Tempe has turned it's attention to smoothing out the construction process of additional Monsoon systems.

Joint planning by MIT and Motorola, for the follow-on *T system has begun. Besides being a quasi-product, i.e., more easily manufactured and therefore available in larger quantities than Monsoon, it is expected to be price/performance competitive with what will then be available in the market place and Id program compatible with Monsoon such that all Id software currently under development will be executable on *T.

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

September 20, 1990:	Monsoon Status Meeting, Cambridge, MA.
November 20, 1990:	MIT Internal Project Review, Cambridge, MA.
November 27-28, 1990:	MIT-Motorola Joint Project Review, Cambridge, MA.
January 7, 1991:	MIT-Motorola *T Planning Meeting, Tempe, AZ.
January 17, 1991:	MIT-Motorola *T Planning Meeting, Tempe, AZ.
March 14, 1991:	Monsoon Software: The Last Stretch meeting, Cambridge, MA.
April 29-30, 1991:	MIT-Motorola Joint Project Review, Cambridge, MA.
May 2, 1991:	MIT-Motorola *T Planning Meeting, Cambridge, MA.

4 Other external collaborations

In the past, our research has benefited from the collaborative efforts of our group with researchers from other institutions. Such interactions continue to be encouraged. During the past year, graduates from the Computation Structures Group have gone on to join efforts at other interested research institutions. These factors coupled with the achieving of several research group milestones during the past year have facilitated an expansion of this participatory and investigative community.

4.1 IBM

At IBM Research, Bob Iannucci and K. Ekanadham have continued to use Id World software in their work. Ekanadham has developed an intermediate language called Kudos that allows him to express hybrid dataflow graphs that can have both sequential and parallel subgraphs. Ekanadham has modified the Id compiler to generate Kudos from program graphs and is generating code for the Empire architecture from Kudos.

4.2 MITRE

Under a MITRE Mission-Oriented Investigation and Experimentation project, Dr. M. C. Michaud has investigated the parallelism inherent in typical signal processing applications independent of any target programmable parallel signal processor. This research used the Id and GITA for exposing inherent algorithmic fine-grain parallelism. MITRE with assistance of the Computation Structures Group has made modifications to the GITA simulator in order to investigate coarse-grain algorithm parallelism as well. First, R. Paul Johnson of CSG and Gregory M. Whittaker of Mitre added a filtering capability to GITA to allow one to view the contribution to parallelism associated with user-selected code-blocks. Secondly, they added a probe function to the Id compiler and GITA that identifies the time-step associated with transmission of a data value in some specific context.

MITRE studied two signal processing algorithms both for their fine-grain parallelism and their coarse-grain parallelism. The two algorithms are the estimate of the autoregressive power spectral density and the QR factorization of a matrix using the modified Gram-Schmidt method. Id and GITA, with the modifications, have been successfully used to identify both fine-grain and coarse-grain algorithm parallelism in these algorithms.

4.3 Berkeley

On leaving MIT after finishing his Ph.D., David Culler has joined the faculty at the University of California at Berkeley. Culler is investigating multi-threaded machine architectures modeled by a threaded abstract machine, TAM, which features compiler directed synchronization, multilevel scheduling and storage management. Using a new back-end for the Id compiler, `threadid` machine code is generated and compiled to be executed on stock hardware with performance comparable to LISP and C. Work is underway to generate native code directly from TAM code. Culler's group has developed run-time systems for MIPS and Sun workstations and the NCUBE MIMD parallel processor.

4.4 Sandia

Sandia is continuing its work in threaded dataflow computing. The Epsilon-2 system is their first demonstration vehicle. The Epsilon-2 processor is operational, and executes threaded dataflow graphs as its native code. The rest of the system (structure unit and network interface) is under construction.

Sandia has also developed the required analysis techniques for sequentializing dataflow graphs, and are targeting these algorithms to the Id compiler for execution first on Epsilon-2, then on future multi-threaded machines.

Sandia is also working with David Culler on the task of running Id applications on Sandia's large NCUBE MIMD parallel processor. This work will offer the first Id implementation on a large parallel machine, and provide a platform for research into resource management and other issues in the context of a large multiprocessor.

Lastly, Sandia is beginning to migrate some of its applications to Id, beginning with a discrete Monte Carlo simulation of low density molecular dynamics. This work will increase the number of Id programs available to architecture and compiler researchers as well as increase the number of programmers exposed to the benefits of Id.

4.5 Los Alamos

Los Alamos has a continuing effort to investigate the feasibility of the dataflow model of computation and functional languages designed for numerical computation. As the dataflow environment matures and prototype hardware becomes available, Los Alamos is focusing its efforts toward developing realistic application codes written in the Id language and targeted toward the Motorola Monsoon machine. The application codes will help to focus the computer science and engineering research, determine the feasibility of the dataflow concept and language, provide performance and scalability data, and drive further development. Towards this end, they have selected a very general Monte Carlo transport code called MCNP as an application that can benefit significantly from a massively parallel MIMD architecture. Their intention is not to rewrite MCNP in its entirety because it includes extensive user features that are unimportant to its computer performance, but rather to rewrite the core of the code that retains its numerical and physical complexities (we will refer to this code as MCNP-ID). Algorithmic changes may be desirable and necessary during the process.

4.5.1 MCNP Description and Significance

The most widely used Monte Carlo transport code in the world today is probably MCNP, developed over the last 30 years at the Los Alamos National Laboratory. MCNP is a general purpose Monte Carlo code for calculating the time-dependent continuous-energy transport of neutrons, photons, and electrons in either single particle or coupled particle mode in general three-dimensional geometries. More than 350 person years have gone into the research, development, programming, documentation, and databases for MCNP. MCNP is heavily used in a variety of applications at Los Alamos and, in addition, the code is in nearly universal use throughout the country and worldwide. In fiscal years 1989 and 1990 alone, MCNP was requested by 16 federal laboratories, 48 industrial companies, 31 universities and technical centers, and 21 foreign organizations. This does not include requests to the Radiation Shielding Information Center in Tennessee, which distributes MCNP as part of its computer code collection. MCNP is the principal radiation transport design tool for the oil well logging industry and the magnetic fusion energy community. Other applications of the code include nuclear reactor design (fission and fusion), criticality safety, radiation shielding, nuclear safeguards, personnel dosimetry, radiography, energy deposition, detector design, particle physics research, materials science, radiotherapy, accelerator target design, climate and global change, astrophysics and aerospace design.

4.5.2 MCNP-ID Capability

The intent of this project is to rewrite the core physical and numerical processes of MCNP in Id. The complexity of the code argues that the development take place in a staged process. Currently, Los Alamos is progressing toward the completion of the first stage. This stage will include the general user-specified geometry, the simple and detailed photonics transport, variance reduction techniques, and statistical tally information. At the completion of this stage, MCNP-ID will be capable of executing some real-world published standard transport problems. During the latter part of the first stage or as a possible follow-on project, they will add neutron particle transport. For computer performance analysis, there are three types of transport problems, classified according to where the computational effort is predominately spent. Specifically, these areas are tracking particles through a complex geometry, collision mechanics and making statistical tally estimates. Los Alamos intends to generate problems exercising the code and dataflow environment in all three categories. A comparison between the FORTRAN MCNP and MCNP-ID will be possible for these problem types.

Initial development is taking place using the Motorola MINT system and the MIT Id World system. As development continues and the MCNP-ID code matures, the project will be moved to actual Monsoon hardware (a one PE system which is currently in-house). Finally, full scale feasibility and scalability studies will be accomplished on a full 8 PE system.

5 Id: general topics

The Id programming language continues to evolve. The reference manual for a new version (Id version 90.0) was released in early fall 1990. Major additions to the language include M-structures, overloading in the type structure, and input/output capabilities. Storage reclamation in a parallel language remains a difficult problem, and for now we are experimenting with annotations for explicit storage release. We have also made substantial progress in developing a high-level, machine independent formal operational semantics for Id.

5.1 Id 90 and M-structures

The most significant addition to Id is "M-structures". Previously, Id had a purely functional core, extended with I-structures. The functional core was both referentially transparent and deterministic. With I-structures, the language loses referential transparency, but remains deterministic.

The introduction of M-structures also introduces non-determinism. This is quite a serious loss, but we have gradually come around to the view that certain programs that exploit non-determinism can be clearer, more parallel and more efficient (as measured by instruction and storage counts), while retaining determinacy at a more macro level. Consider a program to traverse a graph structure, for example to count the number of nodes reachable from a given node. In order to avoid repeated traversals of shared subgraphs and cycles, the program needs to maintain a table of already-visited nodes. In both functional and I-structure programs, insertions and lookups on this table must be performed in a deterministic order, which clutters the program by threading the table in and out of all function calls, and sequentializes the traversal of the graph. If insertions could be performed non-deterministically, the graph may be traversed in parallel. The final outcome (number of nodes reachable) is still deterministic although the order of the insertions is non-deterministic. M-structures permit the expression of such non-deterministic algorithms.

M-structures are data structures with updatable components; however, reads and writes of M-structure cells have built in synchronization. The read and write operations are called *take* and *put*, respectively. An M-structure cell is either in a *full* or *empty* state (initially empty, when the cell is allocated on the heap). A PUT on an empty slot stores a value there and marks it full. A TAKE on an empty slot simply blocks until it is full. When there are multiple TAKEs on a full slot, at most one of them succeeds in retrieving the value; the slot is marked empty again, and the remaining TAKEs stay blocked on that slot. For example, the array increment statement:

```
A![j] = A![j] + 1
```

is guaranteed to be atomic, because after the value is TAKEN (right-hand side), no concurrent computation can read a value from the slot until the incremented value is PUT back (left-hand side).

M-structures certainly introduce new complications in programming (including explicit constructs to sequentialize two program fragments), and introduce the possibilities of subtle bugs (non-determinism, incorrect pairing of TAKES and PUTS, new kinds of deadlock, *etc.*) To date, we have not yet had enough experience with M-structures to articulate a comprehensive methodology for using M-structures. We hope to gain this experience in the coming year.

For his Ph.D. thesis work, Paul Barth has been looking at higher-level encapsulators for managing groups of M-structure operations on complex data structures and on multiple data structures. Explicit use of M-structures in such situations can become very messy in the details of orchestrating the TAKES and PUTS correctly. Barth's proposed "`enter`" construct is an attempt to relieve the programmer of such details.

5.2 Overloading

The Id language permits overloading of identifiers and operators with multiple meanings. For example, the symbol "+" may stand for integer addition or floating point addition according to the context of its use. During the winter of 1990-1991, the Id language syntax and the compiler front-end were extended by Shail Aditya to process such overloading and instance declarations. Thus, the overloaded addition is declared as follows:

```
overload + = *0 -> *0 -> *0;
instance + = plus~int, plus~float;
```

Here, "+" is declared to have a template of a binary operator that can be *instantiated* to integer addition or floating point addition implemented by primitive operations `plus~int` and `plus~float`, respectively. After these declarations, an expression such as `x+2` could be recognized by the type-checker as integer addition and replaced by `x plus~int 2`.

An earlier proposal by Nikhil in 1988 for identifying and resolving such overloaded identifiers in a given context was based on the solution adopted in the programming language Haskell [13]. This scheme added extra parameters to user-defined functions whenever it could not resolve the overloaded identifiers locally. Experimentation and experience with such *implicitly* overloaded functions led us to conclude that this scheme was unacceptably inefficient in practice without a significant gain in flexibility. In spring 1991, Shail Aditya revised the overloading resolution and translation mechanism based on a proposal by Arvind and Nikhil.

In the new scheme, only *explicitly* overloaded identifiers are allowed. An implicitly overloaded function that cannot be fully resolved locally, must get completely resolved in the global context (in the complete program). No overloaded functions are ever allowed to run. This scheme permits a given overloaded function to be resolved in exactly one way and hence is slightly inflexible, but it does not generate any extra parameters and therefore incurs no extra run-time cost. The standard libraries are equipped with a large set of explicitly overloaded operators and functions along with their instances.

The resolution and translation of overloaded identifiers occur in two phases. The type checker is responsible for propagating the local overloading information to successively broader scopes until the implicitly overloaded function gets completely resolved. Many subtleties arise due to interactions with polymorphic instances and incremental type-checking. The type checker ensures that exactly one resolution takes place by temporarily making the overloaded identifier monomorphic in the current scope. As soon as the resolution is determined, the polymorphism of the resolved instance is restored. Finally, global resolution is performed over the complete program and all the resolved identifiers are transformed into their instances.

5.3 Input/Output in Id

Some basic input/output (I/O) capabilities have finally been introduced into the Id language. This does not involve any syntactic or semantic changes to the language; the I/O capabilities are implemented as a standard library.

There were two key technical questions to be addressed. First, in pure functional languages, I/O is typically introduced by considering each top-level program to be a function from an input stream of operating system responses to an output stream of operating system requests. Because of non-strictness, a request can be sent on the output stream before looking for the response in the input stream. There are many problems with this style: many people find it very unnatural, programs can become very messy, having to "plumb" these I/O streams through numerous function calls and returns, and unrelated I/O actions have to be sequentialized into the single input and output streams. It was therefore decided not to follow this model, but rather to have the conventional, imperative, input/output *statement* for each I/O operation.

This brought up the second technical issue: in a parallel language, the order in which I/O statements are executed is unpredictable. How then, for example, to write items to a file in a deterministic order ? Our solution is to extend each I/O procedure with an extra argument (a "trigger") and an extra result (a "status"). The procedure is strict in the trigger, *i.e.,* it does not attempt to perform any I/O until it receives the trigger. Furthermore, the status output is not returned until the I/O operation has completed. By threading the status output of one operation into the trigger of another, I/O operations can be forced to occur in a particular order.

The threading of statuses and triggers again raises the issue of "plumbing", one of our objections to the purely functional style. The differences are that unrelated I/O actions, such as output to two different files, can be threaded separately in our system, but not in the functional style, and that in situations where I/O does not require an ordering (such as rendering lines of a figure onto a screen), we can omit the threading and allow them to happen non-deterministically.

A first cut at an I/O library, loosely modeled on the Unix standard I/O library, has been implemented in GITA, and has been in use since April 1991. We hope to refine the library based on the experience we will gain using it in the next few months, and to extend its functionality. It also needs to be implemented for the Monsoon system.

5.4 Annotations for storage reclamation

James Hicks has continued his work with annotations and compiler analysis for storage reclamation in Id programs. Hicks is developing a theoretical framework based on abstract interpretation that allows the compiler to verify or insert storage reclamation annotations in Id programs. Hicks implemented an analyzer based on the principles of the theoretical framework and a module that uses object lifetime information to insert or verify storage reclamation code.

This analysis and transformation system has proven to be very successful with programs with regular structure. When applied to the SIMPLE[11] benchmark, the compiler can automatically insert storage reclamation code for every temporary matrix and tuple allocated in the program. The resulting code has been run for a thousand iterations on a 1-PE, 1-IS Monsoon machine. This system also has complete success with Wavefront, reclaiming all intermediate matrices. When applied to the Gamteb[8] photon-transport benchmark, the system inserts 23 deallocation statements, whereas the hand-annotated version has 25 deallocates. The compiler fails to find the last two because the programmer made some additional assumptions about the inputs to the program that the compiler cannot make.

However, the compiler insert these two deallocation statements if it also inserts run-time tests to guarantee their validity. This problem can be solved by augmenting the transformation module; the analysis module gathers all the necessary information.

Hicks has also implemented a module that converts heap-managed objects of fixed size to frame-managed objects when the object's lifetime ends upon termination of the nearest enclosing procedure body. The Monsoon Back-End reserves frame slots for the frame-allocated objects, and converts the allocation code to code that generates an I-structure pointer to those frame slots. This transformation eliminates the cost of allocating the object — it is tied with the cost of allocating the frame. The only cost of deallocating the object is the clearing of the presence-bits of the frame slots used by the object, but this cost is paid even if the object is heap-managed. A few experiments with the Gamteb benchmark show that frame-allocation reduced the number of instructions executed by twenty percent. This work with frame-allocation illustrates that lifetime analysis is not sufficient — intelligent decisions about how to transform the code must be made given object lifetime information.

Further work is needed to work out all the details of the theoretical framework and to improve the implementation. The current implementation deviates slightly from the theoretical work and is fairly inefficient. Although in most cases, it is still practical to apply the analysis to fairly large programs such as SIMPLE and Gamteb, in other cases the analysis module causes compile times to grow exorbitantly. This inefficiency should be removed by improving the caching of information and improving the calculation of fixpoints.

5.5 Operational semantics for Id$^-$

Zena Ariola and Arvind have completed the specification of the operational semantics of a subset of Id called Id$^-$, which is Id without comprehensions and M-structures. In order to keep the exposition clear, Id$^-$ contains only lists as an example of general union types. A procedure to translate the abstract syntax of Id$^-$ into Kid, our Kernel language for Id, is given in [1]. This translation procedure also includes the rules for pattern matching.

Next, the operational semantics of Kid is presented in terms of rewrite rules using an innovative rewriting system, which has been named *Contextual Rewriting System (CRS)*. In a CRS, the block construct (*i.e., letrec*) is treated as a first class citizen, and not merely as syntactic sugar for function application. Kid, together with its CRS, accurately captures the sharing of subterms and the operational behavior of I-structures. A notion of *answer* associated with a Kid term is developed. This notion of answer extends the usual notions of normal forms in λ-calculus, and is not directly related to termination issues. A normalizing interpreter for Kid, with respect to this definition of the answer is presented in [2].

Optimizations performed by the Id compiler are formally expressed as source-to-source transformations in Kid. Different notions of equality are defined, such as tree-equivalence and graph-equivalence. These notions, though syntactic, are powerful enough to prove correctness of some of the common optimizations, such as common subexpression elimination. However, to fully characterize the correctness of optimizations, such as algebraic equalities, the notion of *observational congruence* is introduced. Intuitively, observational congruence is a relation which equates all terms producing the same answer.

The last phase of the Id$^-$ compilation process consists of choosing a representation for data-structures and closures. This is presented as a translation of Kid into P-TAC, a language that is even smaller than Kid and closer to a parallel machine language [1]. Moreover, since Kid is a typed language and P-TAC is untyped, collecting information in way that is usable by various run-time system procedures such as garbage collectors, *print* procedures or source-level debuggers is a research issue of some interest.

Work is under progress on the specification of the operational semantics of full Id. This has required extending Kid to deal with M-structures and synchronization barriers, and all algebraic types. Other than that the work involves dealing with the full abstract syntax of Id (see the section on Id compiler in Id).

On the theoretical front, work is under progress to give proper semantics to CRSs. CRSs can be viewed as another formalism for graph rewriting. Ariola, following the work of Wadsworth and J-J levy on approximate reductions [21, 14], is trying to develop a term model for the λ_B-calculus, the CRS version of the λ-calculus. This will provide a useful calculus of infinite terms.

6 Id: compiler and run-time systems for Monsoon

Compiler and run-time systems research has continued to play a central role in our research. With Id language development at a significant point of maturity, group attention is shifting to address compiler and run-time systems development. Language and architecture research during the past year has prompted much compiler and run-time research for Monsoon and other dataflow, P-RISC, and von Neumann style target architectures.

6.1 Status

Several new constructs were added to the Id language syntax and the front-end during winter 1990-1991. These include: `typeconverter`, `typesyn`, `overload`, and `instance` declarations. The semantics of these experimental constructs was under study for a while. They were included in the syntax of the language after demonstrating sufficient usefulness. These constructs help the programs look much cleaner.

Overloading and instance declarations allow the user to explicitly declare overloaded identifiers and their instances which will then be used by the type-checker in its resolution process. Typeconverters are identity functions that simply change the type of a given object. These are useful in low-level system code, where the strong typing of Id can sometimes hinder efficient implementations. Type synonyms allow the user to conveniently abbreviate complex type expressions in a systematic fashion.

During February 1991, we undertook a concerted effort to improve the effectiveness of various phases of the Id compiler. James Hicks and Shail Aditya installed a loop-unrolling module that recognizes `for`-loops and generates unfolded loop bodies without extra loop predicates. Shail also examined the desugaring of list comprehensions and installed a new method of desugaring that yields code with more parallelism than the previous version.

Boon Ang implemented a module based on a schema developed by James Hicks that allows multiple values to be returned from Id procedures without allocating storage. No change is made at the source level, because this would have affected the polymorphism of procedures. The module recognizes procedures that return functional tuples of values, and compiles the procedure with two entry points. If the procedure is called through the standard entry point, then a tuple is allocated and returned. If the procedure is called through the alternate entry point, then the individual values are sent to successive instructions at the return continuation. A full-arity application of a procedure will be translated to a full-arity, multiple-return-value application if the compiler can determine that the procedure has a multiple-value form and only the tuple-components of the result of the application are read. Most of the work in this module consists of propagating tuple construction through `if`, `case` and `loop` constructs and recognizing when the multiple-value `apply` schema can be used, so that tuple allocation at run time can be minimized or eliminated.

As part of his Bachelor's thesis [9], David Choi (supervised by Yuli Zhou and with the help of Jamey Hicks) has implemented an array bound checking optimization module for the Id compiler. To insure a semantically correct implementation of arrays, bounds checking code needs to be inserted before array references, that performs run-time checks to guarantee that each index value is within the bounds of the array. However, it is usually felt that the added security does not justify its cost, since such bounds checking code can incur considerable run-time overhead. For example, the dynamic instruction count increases by 53% for the matrix multiplication program with bounds check code inserted, and the overhead is 80% for the wave-front program.

Choi's optimization consequently tries to optimize bounds checking code to reduce the overhead to an acceptable level, while preserving the semantics of a correct, bounds checked program. The optimization he implemented hoists bounds checks out of for-loops when the array index is linear expression in the loop's induction variable. This is a significant reduction in cost, as the optimized checks are performed only once outside the loop. Frequently the optimized checks become constants and will be removed altogether by later peep-hole optimizations. The optimization turns out to be extremely effective, as arrays are usually referenced in the body of for-loops, which are more over regions of high execution frequency. Experiments show that the overhead for matrix multiplication is reduced to 0.2% with the optimization, and to 0.4& for the wave-front program.

Christine Flood has been writing and running extensive test-suites that verify that all of the constructs and procedures in the Id manual behave as they are supposed to. Flood has developed a system that reads test-vector descriptions, generates and compiles Id code, runs that code on an execution vehicle (MINT or Monsoon hardware) and checks the results. The work has been used to verify that the Id compiler and Monsoon Back End generate correct code for Monsoon. This work has enabled us to correct errors in Monsoon microcode, the Monsoon Back End and in the Id code for the libraries. This work is continuing — about half of the libraries have been tested to date.

6.2 Frame Allocator

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 predictable. During the spring 1991 semester, Derek Chiou has been working on two parallel frame mangers for Monsoon.

The first frame manager was proposed by Jonathan Young who held a post-doc position in our group. Young's scheme minimized contention for global resources by distributing the lock between threads in the processor. Chiou took the skeleton code he had written, filled it in, wrote an initializer for it, and debugged it. Data was collect from that manager. The manager was fairly complicated, however, and did not scale to multiple frame sizes easily. Thus, it was decided to write another frame manager from scratch. This one would pay particular attention to multiple frame sizes. Chiou has completed this frame manager and has successfully run it with the latest version of our heap manager on 1 and 2 processor simulations.

6.3 Heap Allocator

Arun Iyengar is studying ways to efficiently manage heap space (also known as dynamic storage) on multiprocessors. An object which is allocated from the *heap* has an indefinite lifetime. It may still be accessible after the procedure which created the object has terminated. Since Id programs use dynamic storage quite frequently, the heap allocator has a significant effect on system performance.

Simulation experiments using several different dynamic storage allocators have been performed. Quick fit displayed the lowest average instruction count and shortest critical path length of all algorithms we simulated. A first fit system using boundary tags displayed the second lowest total instruction count. However, it can only satisfy a single allocation request at a time. Furthermore, it requires an extra word for each block of storage.

Both the quick fit system and the first fit system using boundary tags on Monsoon have been implemented. They are both written in Id. The first fit system keeps all free blocks on a single free list. The allocator always returns the closest block to the head of the free list which is large enough. This allocator has little parallelism because only one allocation request can be satisfied at a time.

The quick fit system maintains separate free lists for small sized blocks which are typically the ones requested most frequently. This algorithm has significantly more parallelism than the first fit system because different free lists can be searched concurrently.

We are trying to improve upon quick fit by managing large free blocks in different ways. We have come up with two new algorithms, ultrafast fit and superfast fit, which have sophisticated methods for dealing with large free blocks. In order to obtain realistic comparisons of quick fit, ultrafast fit, and superfast fit, we are collecting information about request distributions from Id benchmarks.

6.4 Id World programming environment

Support continues for Id World, the Id programming and experimentation environment. Paul Johnson released versions 4.5 and 5.0 of Id World for external distribution. Released in February, V4.5 features Id 89 syntax, and I/O. Released in March, V5.0 features Id 90 syntax.

Id World images on development machines: TI Explorer, Symbolics Lisp Machines, and Sun Workstations, are kept consistent through the use of a common patch system. Releases are maintained by the use of version component control.

Utilities for accurate reading and printing of double precision floating point numbers have been implemented. The floating point reader[10] has been implemented in Common Lisp and Id 90, the floating point printer[17] in Common Lisp. Both the reader and printer have been compared to the native double precision utilities in Symbolics, TI Explorer, Lucid and Allegro Common Lisps. The only differences were seen between the floating point printers, the accurate printer was occasionally worse (longer print representation) than the Symbolic's printer. The accurate printer was better (shorter print representation or more accurate) than TI Explorer, Lucid and Allegro Common Lisp printers.

6.5 Compiling Id for sequential von Neumann machines

Bradley C. Kuszmaul has been working on retargeting the existing Id compiler for stock hardware, such as conventional Unix workstations.

Dataflow graphs produced by the Id Compiler are translated into parallel control flow graphs based on the P-RISC abstract machine model. The major effort is in the analysis and transformation of the parallel control flow graph to reduce the cost of running Id programs on hardware that has no specialized synchronization primitives. The work done by his compiler includes strictness analysis, subscript analysis, identification of threads, transformations to lengthen threads and reduce synchronizations, and peephole optimizations. Finally these graphs are used to generate assembly code for a number of machines including Culler's thread abstract machine [12] (Kuszmaul wrote this back-end, in consultation with David Culler's research group at Berkeley), Madhu Sharma's proposed P-RISC machine (currently run on a simulator), and SPARC and 88K (both of which are currently being debugged by Kuszmaul.)

This work should help us understand many aspects of high performance implementations of functional languages and high performance parallel computing in general. We expect this work to be directly applicable to the *T project (since, *e.g.,* longer threads improves performance on *T), and we expect this project to lead to better compilation techniques for Monsoon. We expect that this work will help us determine how important it is to have specialized synchronization hardware in order to get good performance on Id programs. As part of the Ph.D. work, Kuszmaul plans to run experiments to determine how important the various optimization techniques are, and to determine how much of the synchronization can not be removed by a compiler. Therefore, it will tell us how cheap synchronization must be.

This work should also shed light on the differences in implementation requirements between non-strict, lenient languages such as Id, and non-strict, lazy languages such as Miranda.

We expect to release a version of Id World using this new compiler by mid-summer 1991. This implementation should substantially increase the availability of Id World to researchers who do not have dataflow machines, but who want reasonably good performance for their Id programs.

6.6 A new standalone, portable Id compiler

Since the summer of 1990, Rishiyur Nikhil and Harry Hochheiser have been working on a new, standalone, portable Id compiler. The goals of the project are:

- To produce a standalone compiler that will run on ordinary workstations. The current Id compiler is written in Common Lisp and is very slow; these factors have been a significant obstacle in our ability to distribute Id externally and to build a community of Id programmers and researchers.
- To use and evolve the P-RISC abstract machine as an intermediate language so that it is possible to generate code for sequential and parallel machines based on conventional microprocessors. We now believe that it is possible to generate code for conventional parallel machines that is competitive with other approaches to programming such machines.

We are bootstrapping our way towards the first objective as follows: we have designed an intermediate language called Schezoid which uses Scheme notation, but has Id's non-strict semantics. The compiler is written in this language, and also accepts Schezoid as its source language. During the initial development, we run our compiler under a Scheme implementation; later, it should be able to compile itself, after which it should be a standalone compiler independent of Scheme. The fact that it does not accept Id source yet is not important— a translator from Id to Schezoid is trivial (a YACC prototype has already been written, but has not yet been polished or integrated with the rest of the compiler).

The compiler translates source code first into "P-RISC graphs", a parallel flowgraph language. We have developed numerous analyses and optimizations on P-RISC graphs, some of which are extensions (for parallelism) of conventional flowgraph analyses. The goal of the optimizations is to reduce instruction counts and to statically schedule threads wherever possible. P-RISC graphs are then translated into code for the target machine.

Currently we have only one back end that translates into C, which is then processed by a C compiler. While this gives us great portability, performance suffers because the nature of C code that we generate defeats the optimizations normally performed by C compilers. Our uniprocessor implementation currently runs about 5 to 10 times slower than the STGM compiler for lazy languages (from Glasgow University). We believe that this gap can easily be closed and overtaken by writing new back ends that go directly to assembler for various machines.

Another aspect of this compiler is that it will be quite easy to link to procedures and libraries written in other languages, such as FORTRAN scientific libraries, X-windows in C, *etc.*

We plan to proceed in three stages: (1) Evolve the compiler till it is a robust, standalone compiler for uniprocessor workstations, improving the back end till it is competitive with other uniprocessor implementations of non-strict functional languages; (2) produce back ends for shared memory multiprocessors (which are expected to be numerous and easily available to most researchers in the near future), and (3) produce back ends for distributed memory MIMD machines such as hypercubes, *T, etc We expect to achieve the first stage by the end of 1991. Along the way, it is possible that we will also hook it up to different front ends such as Haskell and Multilisp.

6.7 Id compiler in Id

At the beginning of April 1991, we have started the project of writing an Id compiler in Id (the current compiler is written in Common Lisp). There are many obvious advantages of having an Id compiler written in the same language, including that this enables the compiler to be run in parallel on the dataflow architecture itself. The compiler will be the largest program written in Id so far and will be a good test for Id's software engineering capabilities.

At this point, we are well prepared for the project. This is partly due to valuable experiences gained with the Id compiler written in Common Lisp [19], and partly due to an effort in formalizing various stages of the compilation process as well as the operational semantics of Id^-, a subset of Id [1]. We shall be implementing the full Id language [15], with list and array comprehensions, pattern matching, explicit sequencing using "barriers", I-structures, M-structures and general algebraic types.

The compiler will be organized into the usual three major components: the front-end which deals with parsing, desugaring and type-checking; the middle-end which comprises many machine-independent optimization modules; and the back-end which is responsible for machine-dependent optimizations and generation of machine code. Currently, documentation on the front-end modules are being prepared (Shail Aditya, Alex Caro, Stephen Glim, and Yuli Zhou), which describes Id's abstract syntax as well as the algorithms (translating comprehensions, scope analysis, removal of patterns) that work on abstract Id syntax trees. The desugared language will be translated into Kid (for Kernel Id), a much simpler language than Id, on which type-checking will be done. It is hoped that these documents will give precise instructions to coding the front-end, which will take place in early June and is expected to be completed by the end of June or early July.

7 Monsoon hardware development

There are two execution vehicles for Monsoon code; the Monsoon hardware and the MINT interpreter. Work on the Monsoon hardware has centered on the production of working boards and the integration of those boards into systems. Work on MINT has centered on extending it to support the complete IS2 Monsoon instruction set and adding more instrumentation.

7.1 Processing element and I-structure memory board

As reported in the foregoing, Motorola completed the assembly and debug and delivered to MIT first build of Monsoon processing elements and I-structure memory boards. This hardware has been used to support the software development effort. This use has uncovered a few small design errors in the PE and in the IS. All of these design errors have been remedied by simple engineering changes (ECO's). These ECO's have been incorporated into all existing PE's and IS's and they will be incorporated into all future PE's and IS's.

7.2 Interconnection network and system integration

During the last year, Motorola completed the assembly and debug of a build of switch boards and completed a build of network cables. Some problems occurred in the construction of the network cables. In particular, testing at Motorola and MIT determined that the vendor's (Rogers') original design connector was not reliable. Rogers remedied these problems by modifying the connector design.

Motorola also completed the construction of the first enclosure for an 8-PE, 8-IS system. This enclosure includes four card cages, four Unix host computers, four sets of power supplies, and two air conditioners.

Using these components, Motorola assembled and debugged the first 8-PE, 8-IS system. As part of this work Mike Beckerle and Ken Traub developed a set of software tools for verifying the network integrity and diagnosing network faults. They also integrated these and other tools into a package for verifying the integrity of the system and for diagnosing system errors. The system was delivered to MIT in May 1991.

7.3 MINT

In a cooperative effort with MIT, Motorola completed the MINT implementation of the full IS2 Monsoon instruction set[18]. MINT is a software interpreter that emulates instruction execution on single or multiple processor Monsoon configurations[6]. Much of the MINT emulator is generated automatically via tools that ensure exact compatibility with the Monsoon hardware and microcode[5].

In our efforts to migrate experimental users from the GITA interpreter (which makes use of the abstract TTDA instruction set) to the MINT interpreter (Monsoon instruction set), we have added substantial instrumentation and a statistics viewer to this tool. Detailed statistics are now available for run-time system activities: all code-block invocations and terminations, frame storage allocation and reclamation, I-structure allocation and reclamation. The statistics viewer allows graphical display of a large variety of statistics in the X-windows environment.

Stephen Brobst has extended the Monsoon instruction set in MINT with fictitious operators for storage management[7]. These operators allow greater flexibility in studying the execution behavior of programs on the Monsoon architecture by separating instructions for program execution from run-time system instructions. Moreover, the "fictitious" operators allow for experimentation with a variety of schemes for frame and structure allocations. Derek Chiou has also extended the Monsoon instruction set in MINT with additional capabilities for synchronization instructions using the "TAKE" primitive. Additional scheduling strategies have also been integrated into the current version of MINT.

8 Applications and performance measurements

In February 1991, a significant effort was made to run Id applications on Monsoon hardware to determine their performance. James Hicks, Christine Flood, Shail Aditya, Alejandro Caro, Paul Barth, and Arun Iyengar worked on six of our Id applications to get them to run on the hardware. In some cases the intent was to show the best performance or utilization of the machine possible, in others, the intent was to show that applications with data sets of non-trivial size could be run.

The six applications were: a matrix multiplication benchmark, the wavefront example, the Gamteb[8] benchmark, the Paraffins benchmark[3], a simulated-annealing approach to the Traveling Salesman Problem and the SIMPLE hydrodynamics benchmark[11]. All of these applications were written in Id 90 with some annotations for storage management.

8.1 Matrix Multiply

This benchmark creates two matrices of size $n \times n$, multiplies them, and returns the sum of the elements of the product matrix as its result. The matrices all contain double-precision floating point numbers. The matrix multiply is written as a straightforward triply nested loop. The innermost loops of the procedure that created the matrices, the matrix multiply procedure, and the procedure that summed the elements of the product were all unfolded 10 times by the compiler to ameliorate the overhead of loop iteration.

Running this benchmark with $n = 500$ took 4 minutes, 10 seconds. The following table contains some of the pertinent statistics for the run of size 500.

FF/QF	insts. fired	instructions	CPI	min:sec
FF	1.74×10^9	2.15×10^9	1.16	4:10

where *instructions fired* is the number of instructions issued minus the number of bubbles, and CPI (cycles per instruction) is the ratio of the number of instructions issued to total cycles.

The column headed *FF/QF* shows which heap storage manager, `first-fit` or `quick-fit`, was used. The `quick-fit` manager uses several free-lists so it suffers less from contention than first-fit does.

In this example, 14 percent of the cycles were idle. Most of these idle cycles are because of contention on the bus between the end of the processor pipeline and the token queues and network interface, which causes an idle to occur whenever a token is issued to the network and no token takes the direct recirculate path in the processor. All fetches from I-structures in dataflow code have this property, and 10 percent of the instructions in matrix multiply were I-fetches, accounting for most of the idle cycles.

For comparison, the same program in C operating on matrices of double floats took 8 minutes, 25 seconds with $n = 500$ on an unloaded SPARCstation 1+.

This benchmark was run on several problem sizes and determined that it executes 17 instructions (counting bubbles) in each iteration of the inner-product computation. It takes 31 instructions in each iteration of the inner-product computation if the loop is not unfolded, and in the first version of the Monsoon Back End, that loop took 57 instructions. The reduction from 57 to 31 instructions in the inner loop was accomplished by moving from instruction subset 1 to the complete instruction set.

8.2 Wavefront

Wavefront is a successive over-relaxation problem defines a sequence of matrices X^k such that for each iteration k, X^{k+1} is defined in terms of a matrix X^k, where

$$X^{k+1}_{i,j} = \frac{X^{k+1}_{i-1,j} + X^{k+1}_{i,j-1} + X^{k+1}_{i-1,j-1} + X^{k}_{i,j}}{5}$$

and X^0 is given as an initial parameter and $1 \leq k < m$. The mth matrix is returned as a result.

Wavefront has the property that all of the elements along diagonals of a matrix may be computed in parallel, as soon as the preceding diagonal has been computed. This parallelism is impossible to expose when written in standard FORTRAN. Furthermore, successive relaxations may execute in parallel.

Here are the statistics for a run of 144 iterations of a 500×500 wavefront. The loop bound on the outer loop of `multiwave` was set to 12, which dictated the problem size (we had to have enough heap for 13 copies of the matrix). On a 1-PE, 1-IS configuration, we have 4 M words of heap storage available.

FF/QF	insts. fired	instructions	CPI	min:sec
FF	2.1×10^9	2.8×10^9	1.08	5:00

Wavefront, like matrix multiply, had a very low percentage of idle cycles — about 8 percent. Again, most of these idles were due to I-fetch requests issued across the network, meaning that wavefront is almost completely utilizing the machine.

8.3 Gamteb

Gamteb[8] statistically simulates the trajectory of particles (photons) through a carbon rod. Each particle is statistically *weighted* to emphasize particles that are in right-most cells

The rod is divided into 9 cells and has 11 surfaces. The surfaces are number 1 through 11, with surface 1 being the cylindrical surface, 2 being the face where particles enter the simulation, 11 being the other face of the cylinder, and 3 through 10 being the interfaces between the cells. Another common formulation of this problem has only two cells and 4 surfaces.

The simulation considers n particles independently, where typical problem sizes are 40 thousand to several million particles. Particles all enter the simulation through surface 1, and may exit the simulation in one of 4 ways: *escaping* through surface 1, *back scattering* through surface 2, *transmitting* through surface 11, or *dying* due to lack of statistical significance. The result is three histograms of the energies of the particles that exited, plus counts of particles that underwent various processes.

This program is intensive in storage. It operates on particles and counts functionally, so whenever a new particle or count of events is needed, a new nine-tuple is allocated. This code has been hand annotated with some storage reclamation pragmas so that the compiler will insert calls to deallocate storage. In some measurements we did this year using code block coloring, we determined that roughly half of the useful cycles were spent in the storage manager when running Gamteb,

The following table gives details of several runs of Gamteb on Monsoon.

particles	splits	bound	FF/QF	insts. fired	instructions	CPI	hour:min:sec
$40,000$	$280,000$	10	FF	4.68×10^9	6.17×10^9	1.7	17:13
$40,000$	$280,000$	5	QF	3.77×10^9	5.08×10^9	1.3	11:12
10^6	7.2×10^6	10	FF	—	—	—	7:13:20

The column headed *bound* gives the loop bound of the outer loop — this controls how many particles are simulated in parallel. The amount of parallelism that can be exposed is limited by the number of activation frames available. The quick-fit storage manager used larger frames, so we could not expose as much parallelism at this level for the second run.

The third row of figures contains "—" in three columns because the hardware statistics counters overflowed and no provisions were made to periodically sample the counters.

8.4 Paraffins

The Paraffins benchmark[3] consists of enumerating all of the distinct isomers of each paraffin of size up to n. Paraffins are molecules with chemical formula C_nH_{2n+2}, where C and H stand for carbon and hydrogen atoms, respectively, and $n > 0$.

Here are the results of two runs of the paraffins benchmark. In both cases we were counting the number of paraffins of each size up to 19, inclusive. The only difference between the runs is the heap manager used. This demonstrates that the additional paralelism in quick-fit decreases contention and increases processor utilization.

FF/QF	insts. fired	instructions	CPI	min:sec
FF	165×10^6	216×10^6	2.3	0:50
QF	146×10^6	192×10^6	1.6	0:31

8.5 Traveling Salesman Problem

This benchmark consists of finding an approximate solution to the Traveling Salesman Problem using simulated annealing. It is coded in a non-deterministic style in Id in order to gain parallelism. This program uses the `manager` construct to implement the simulated annealing.

The problem we ran consisted of 500 cities, 1000 temperatures and 50 swaps per temperature.

FF/QF	insts. fired	instructions	CPI	min:sec
FF	87×10^6	115×10^6	1.3	0:15

8.6 SIMPLE

This application is a hydrodynamics and heat conduction simulation program known as the SIMPLE code[11]. The SIMPLE document, along with the associated FORTRAN program, was developed as a benchmark (unclassified) to evaluate various high performance machines and compilers. Though SIMPLE is supposed to reflect some "real applications", it is contrived to reflect a more complex mix of numerical methods than the usual problems in that class.

It uses a Lagrangian formulation of equations to simulate the behavior of a fluid in a sphere. Nodes are mapped onto a 2-dimensional logical grid where grid points have coordinates (k, l) for $k_{min} \leq K \leq k_{max}, l_{min} \leq l \leq l_{max}$. The product, $k_{max} l_{max}$, is the *grid size* of the problem.

The following table summarizes our results of running KT SIMPLE for several problem sizes. KT SIMPLE is a fairly straightforward translation of the FORTRAN SIMPLE code into Id. Computation Structures Group has been studying this particular version for many years, and it has been updated from Id to Id Nouveau to Id 88 to Id 90.

size	iters	FF/QF	insts. fired	instructions	CPI	hour:min:sec
100	1	FF	105×10^6	138×10^6	1.4	0:0:19
100	1000	FF	—	—	—	4:48:00
200	4	FF	1.67×10^9	2.18×10^9	1.4	0:6:05

This version of the program consisted entirely of 1-bounded loops. It was run to prove that we could run a problem of significant size, not to show how good our performance could be. We will now tune the loop-bounds in order to expose enough parallelism to achieve full utilization on Monsoon. Again, the three entries containing "—" indicate that the hardware statistics counters overflowed.

8.7 Parallel B-tree algorithms in Id

Arun Iyengar is implementing parallel B-tree algorithms in Id. A true B-tree contains data in internal nodes. By contrast, a B+ tree stores all data elements in the leaves. We are studying both B-trees and B+ trees. The algorithm which we are using for B+ trees utilizes the B-link structure initially presented by Lehman and Yao. A number of people have found B-link algorithms to have the highest throughput of any class of parallel B-tree algorithms. We are also implementing the Bayer-Schkolnick and Mond-Raz algorithms on B-trees containing data in internal nodes of the tree.

9 Plans for post-Monsoon architecture research

Since January 1991, ideas on a new parallel architecture have been evolving in our group. This new architecture is called "*T" (pronounced "start"), and is a logical successor to the Monsoon project which will soon draw to a close.

9.1 *T: A synthesis of dataflow and von Neumann machines

Our experience with the Tagged Token Dataflow Architecture (TTDA), Monsoon and P-RISC over the last decade or so has crystallized into the following observations.

In order to build parallel machines that are scalable both physically and economically, we must face the fact that inter-node *latency* in the machine will grow with machine size, at least by a factor of log(N), where N is the number of nodes in the machine. Thus, access to a non-local datum in a parallel machine may take tens to hundreds of cycles, or more. If we are to maintain effective utilization of the machine, a processor must perform some other useful work instead of idling during such a remote access. This requires that the processor be multiplexed amongst many threads, and that remote accesses must be performed as *split transactions*, *i.e.,* a request and its response should be treated as two separate communication events across the machine. If we follow this argument a step further, we see that a communication entering a node will arrive at some relatively unpredictable time, and that we need some means of identifying the thread that is waiting for this communication. This is, in fact, a synchronization event.

Thus, the following picture emerges. In a parallel machine, the way to deal with long inter-node latencies is exactly the way to deal with synchronization. A program must be compiled with sufficient *parallel slackness*[1] (or "excess parallelism") so that every processor has a pool of threads instead of a single thread, and some threads are always likely to be ready to run. Each processor must be able to multiplex itself efficiently amongst these threads. All communications should be split transactions, in which an issuing processor does not block to await a response, and a receiving processor can efficiently identify and enable the thread that awaits an incoming communication. For a more thorough explication of this argument, please refer to [4].

[1] The term is apparently due to Valiant [20].

Previous dataflow architectures (TTDA, Monsoon) have always had these capabilities; however, they were deficient in certain other respects:

- Poor single-thread performance: The interleaved pipelines of TTDA and Monsoon meant that instructions in a particular thread can enter the pipeline at most once every s clocks, where s is the number of pipeline stages (on Monsoon, $s = 8$); thus, in single-thread performance, they cannot compete with modern microprocessors. Single-thread performance is important for very heavily used critical sections.
- Poor control over scheduling: In the TTDA, there was no control over the order in which instructions entered the pipeline. Monsoon is slightly better, in that unbroken threads of instructions can retain a pipeline slot. However, in Monsoon there is no control on scheduling across threads. Such control over scheduling is important for resource management, for locality, *etc.*
- Economic viability: since Monsoon is a custom processor, we cannot afford the level of technology used by high volume commercial microprocessors. Further, being a custom processor, we have to implement *all* software from scratch: operating systems, I/O, libraries, graphics, *etc.*

The proposed *T architecture learns from all these lessons. The *T achieves the best of both worlds (single-threading, scheduling control and compatibility of conventional microprocessors, and multi-threading and cheap synchronization of dataflow processors) by embedding an off-the-shelf, modern RISC microprocessor within a node architecture that provides the multi-threading capability of dataflow machines.

The *T abstract model for a node in a parallel machine is shown in Figure 1. Although the memory of the machine is physically distributed amongst all the nodes, we assume a single global address space, *i.e.,* the local memory in a node implements a piece of a single address space.

The Data Processor is a conventional RISC microprocessor, and executes sequential threads supplied to it by the Start Processor, which can be regarded as a co-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 for that node (another co-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

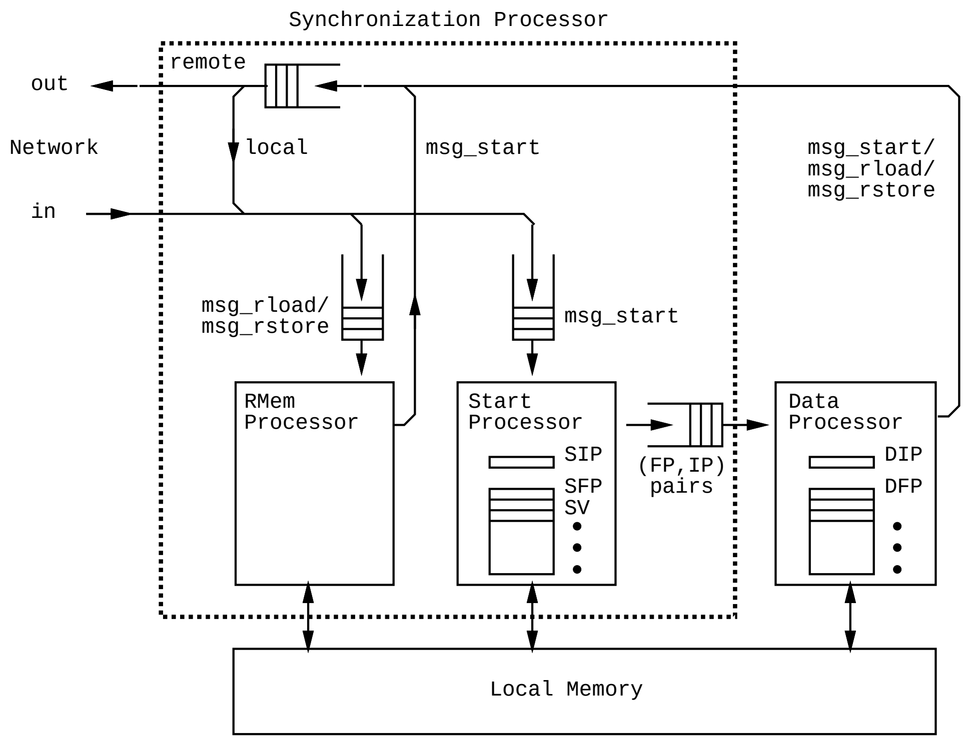

Figure 1: A node in the *T abstract machine

and feeds the named thread to the Data Processor, which executes it after completing the current thread that it is executing.

In this way, as in previous dataflow machines, all computation is event driven— there is no busy waiting for events, be they synchronization events, remote memory loads, or whatever. A detailed exposition may be found in [16]

Compiling for a *T machine is a direct extension of our existing compiler technology. The new compiler projects described in Sections 6.5 and 6.6 are already in a good position to produce code for *T. Our preliminary studies indicate that we can build a *T multiprocessor along these lines that will not only outperform *any* other MIMD machines currently on the market or currently being built (including Intel's Touchstone machines), but will also be just as economically viable. We are thus very excited about *T and hope to establish it as our architecture project that will follow the Monsoon project.

9.2 Prototype work

In the spring, Chris Joerg and Dana Henry have designed and documented the architecture for a prototype *T multiprocessor. This prototype is meant to develop architectural expertise and provide a fast software development platform for the final *T implementation. The prototype uses Motorola 88200 processors and a Network Interface Chip (under design). It borrows Monsoon's network (the Link and PaRC chips). Figure 2 shows the top level architecture of a *T node. One 88200 is used as a Data processor and one as a Synchronization

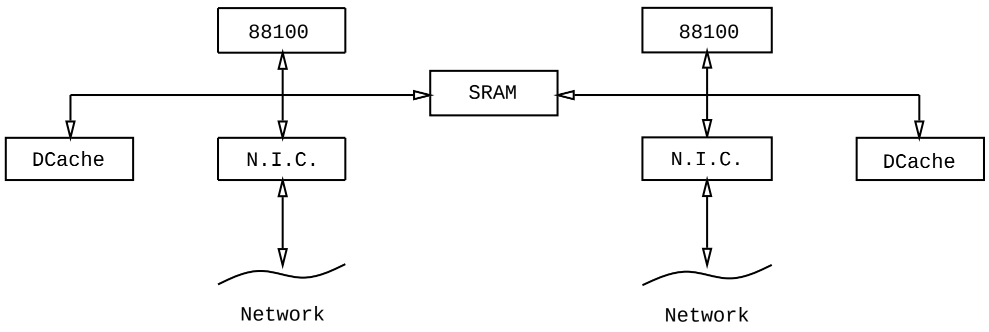

Figure 2: *T prototype

processor. Both are attached to the network via a Network Interface Chip which emulates one of 88200's data caches. The data processor sends messages directly through the N.I.C. Messages are received via another N.I.C. by the synchronization processor. Within a single node, Data and Synchronization processor communication is sped up by a dual ported SRAM which contains procedure frames and thread queue.

10 The AGNA Parallel Persistent Programming Language

In the last year, substantial progress has been made on AGNA. The first notable point is the acquisition of the name "AGNA", which stands for "AGNA's Got No Acronym". AGNA can be regarded as an "object database system".

AGNA is a programming/database system with *expressive power*:

- The user can declare types for complex objects and database-like mappings between them (one to many, many to many, with or without inverses, *etc.*). This permits expression of much more complex database structures than is possible in traditional relational databases.
- The user has the power of a full programming language (including recursion and higher-order functions) to create and manipulate such objects.
- The language includes "list comprehensions" (a popular construct in modern functional languages), which subsumes SQL, today's standard database language for associative (or "declarative") access to data.
- The management of the persistence of objects (which is what makes it a database) is automatic. Any object of any type reachable from the top-level persistent environment is automatically persistent.

- The update language is also declarative: an update transaction is a *specification* of a unique "next state" of the database in terms of the current state.

- The entire language is an implicitly parallel language, in the tradition of dataflow languages.

AGNA exploits compiler optimizations and parallelism heavily to achieve good performance:

- The persistent heap is segmented to group together objects of the same type, and hash and B-tree indexes are constructed for invertible fields. This information is used in the AGNA transaction compiler to build efficient query plans for list comprehensions.

- The optimized code is compiled into P-RISC graphs, *i.e.,* code for the P-RISC abstract machine. The coarse-grain parallelism in the code is used to distribute the transaction over the nodes of an MIMD parallel machine. The fine-grain parallelism in the code is used, dataflow style, to absorb the various latencies of the parallel machine: communication, synchronization and disk I/O.

AGNA currently runs on two platforms: networks of Sun workstations, and a 32 node Intel Hypercube with parallel disks at the University of Wisconsin, Madison (our thanks to Prof. David DeWitt at Wisconsin for permitting us to use it). Preliminary benchmark results indicate that:

- On a single SPARCstation, AGNA runs from 1 to 2 times more slowly than Ingres, a state of the art commercial relational database system.

- On the Wisconsin Intel Hypercube, AGNA runs from 2 to 5 times more slowly than GAMMA, an experimental parallel relational database system built at Wisconsin.

These results are very encouraging, especially considering the more general expressive power of AGNA and the fact that we have not had much time yet to tune the system (we expect considerable performance improvement in the future).

The AGNA work has resulted in two conference publications, at the IEEE Conference on Data Engineering, Kobe, Japan, April 1991, and in the Conference on Functional Programming and Computer Architecture, Cambridge, MA, August 1991.

Publications

Aditya, S. and Nikhil, R.S. "Incremental Polymorphism," To appear in *Proceedings on Functional Programming and Computer Architecture*, Cambridge, MA, August 28-30, 1991. Also: Computation Structures Group Memo 329, March 1991.

Ariola, Z.M. "Notes on the Confluence Property of Term Rewriting Systems and the Lambda-Calculus," Computation Structures Group Memo 321, MIT Laboratory for Computer Science, Cambridge, MA, December 1990.

Ariola, Z.M. and Arvind "A Syntactic Approach to Program Transformation," Computation Structures Group Memo 322, MIT Laboratory for Computer Science, Cambridge, MA, December 1990.

Ariola, Z.M. and Arvind "Compilation of Id$^-$, a subset of Id," Computation Structures Group Memo 315, MIT Laboratory for Computer Science, Cambridge, MA, July 1990.

Ariola, Z.M. and Arvind "Contextual Rewriting System," Computation Structures Group Memo 323, MIT Laboratory for Computer Science, Cambridge, MA, December 1990.

Arvind and Brobst, S.A. "The Evolution of Dataflow Architectures from Static Dataflow to P-RISC," In *Proceedings of the Workshop on Massive Parallelism: Hardware, Programming and Applications*, Amalfi, Italy, October 1989. Academic Press 1990. Also: Computation Structures Group Memo 316, August 1990.

Barth, P.S., Nikhil, R.S. and Arvind "M-Structures: Extending a Parallel, Non-strict, Functional Language with State," To appear in *Proceedings on Functional Programming and Computer Architecture*, Cambridge, MA, August 28-30, 1991. Also: Computation Structures Group Memo 327, March 1991.

Boughton, G.A. "Multiprocessor Packaging," Computation Structures Group Memo 324, MIT Laboratory for Computer Science, Cambridge, MA, December 1990.

Culler, D.E. and Papadopoulos, G.M. "The Explicit Token Store," In the *Journal of Parallel and Distributed Computing*, January 1991. Also: Computation Structures Group Memo 312, June 1990.

Glim S., Ang, B.S., Caro, A., and Shaw, A. "Introduction to the Id Compiler," Computation Structures Group Memo 328, MIT Laboratory for Computer Science, Cambridge, MA, April 1991.

Heytens, M.L. and Nikhil R.S. "List Comprehensions in AGNA, a Parallel Persistent Object System," To appear in *Proceedings on Functional Programming and Computer Architecture*, Cambridge, MA, August 28-30, 1991. Also: Computation Structures Group Memo 326, March 1991.

Kuszmaul, B.C. "A Glitch in the Theory of Delay-Insensitive Circuits," In *Proceedings of the 1990 ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (Tau '90)*, Vancouver, British Columbia, Canada, August 14-17, 1990.

Kuszmaul, B.C. "Fast, Deterministic Routing, on Hypercubes, Using Small Buffers," In *IEEE Transactions on Computers*, Vol, 39, No. 11, ppgs. 1390-1393, November 1990.

Nikhil, R.S. "Notes on Automatic Parallelization," Computation Structures Group Memo 314, MIT Laboratory for Computer Science, Cambridge, MA, July 1990.

Nikhil, R.S. "The Parallel Programming Language Id and its Compilation for Parallel Machines," In *Proceedings of the Workshop on Massive Parallelism: Hardware, Programming and Applications*, Amalfi, Italy, October, 1989. Academic Press 1990. Also: Computation Structures Group Memo 313, July 1990.

Nikhil R.S. and Heytens, M.L. "Exploiting Parallelism in the Implementation of AGNA, A Persistent Programming System," In *Proceedings of the Seventh International Conference* *on Data Engineering*, Kobe, Japan, April 8-12, 1991. Also: Computation Structures Group Memo 320, December 1990.

Papadopoulos G.M., Nikhil, R.S. and Arvind "*T: A Killer Micro for a Brave New World," Computation Structures Group Memo 326, MIT Laboratory for Computer Science, Cambridge, MA, January 1991.

Papadopoulos, G.M. "The Explicit Token Store," In *Journal of Parallel and Distributed Computing*, December 1990.

Papadopoulos, G.M. and Traub, K.R. "Multithreading: A Revisionist View of Dataflow Architectures," In *The 18th Annual International Symposium on Computer Architecture*, Toronto, Canada, May 1991. Also: Computation Structures Group Memo 330, March 1991 and Motorola Technical Report MCRC-TR-10, March 1991.

Traub, K.R., Hicks, J., and Aditya, S. "A Dataflow Compiler Substrate," Computation Structures Group Memo 261-1, MIT Laboratory for Computer Science, Cambridge, MA, March 1991.

Young, J. "Context Management in the Id Run Time System," Computation Structures Group Memo 319, MIT Laboratory for Computer Science, Cambridge, MA, September 1990.

Young, J. "Tests for Monsoon Instruction Subset 1," Computation Structures Group Memo 307-1, MIT Laboratory for Computer Science, Cambridge, MA, September 1990.

Young, J. "Two-Phase Transactions," Computation Structures Group Memo 318, MIT Laboratory for Computer Science, Cambridge, MA, September 1990.

Theses Completed

S.B.

Choi, David J. "Optimizing Array Bound Checking in a Dataflow Computer," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1991.

Perez, Elba G. "Mathematical Library Functions for the Id Programming Language," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1991.

S.M.

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, November 1990.

Chang, Alice A. "A CISC to RISC Optimizing Compiler," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1991.

Henry, Dana S. "Specification and Verification of Real-time Constraints in Coarse Dataflow," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, July 1990.

Hochheiser, Harry S. "A Schezoid Compiler for P-RISC," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1991.

Theses in Progress

Barth, Paul "Atomic Data Structures for an Implicitly Parallel Language," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected August 1991.

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 January 1992.

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 1991.

Chiou, Derek T. "Frame Memory Management for Monsoon," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected August 1991.

Heytens, Michael L., "The Design and Implementation of a Parallel Persistent Language System," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected August 1991.

Hicks Jr., James E. Jr., "Compiler Directed Storage Reclamation by Object Lifetime Analysis," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected August 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 1991.

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

Shaw, Andrew "Compiling Data Parallel Languages for MIMD Parallel Computers," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA. Expected September 1991.

Lectures

Arvind. "Id World Implicit Parallel Programming," CRAY-Motorola Meeting, Chippewa Falls, Wisconsin, July 31, 1990; Summer School in Parallel Processing, Ascona, Switzerland, August 6, 1990.

Arvind. "Project Dataflow," Motorola visit to MIT, August 2, 1990.

Arvind. "Does Programming Parallel Machines Have to be Painful?" Summer School in Parallel Processing, Ascona, Switzerland, Ascona, Switzerland, August 6, 1990; Electrotechnical Laboratory, Tsukuba, Japan, August 13, 1990.

Arvind. “Parallelism in Id Programs,” Summer School in Parallel Processing, Ascona, Switzerland, August 7, 1990.

Arvind. “Dataflow Architectures Monsoon/ETS Model,” Summer School in Parallel Processing, Ascona, Switzerland, August 7, 1990; Indian Institute of Science, Bangalore, India, August 28, 1990; Summer Institute in Parallel Computing, Argonne National Laboratories, Argonne, Illinois, September 14, 1990.

Arvind. “Implicit Parallel Programming,” Symposium on Biological and Artificial Intelligence Systems, Rome, Italy, September 3, 1990; Summer Institute in Parallel Computing, Argonne National Laboratories, Argonne, Illinois, September 14, 1990; University of Wisconsin, Madison, Wisconsin, October 18, 1990; Distinguished Lecturer’s Series, Department of Computer Science, Nicholls State University, Thibodaux, Louisiana, January 22, 1991; Distinguished Lecturer’s Series, Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, Louisiana, January 23, 1991.

Arvind. “Advances in Dataflow Architectures: The Monsoon Project,” Distinguished Lecturer’s Series, University of Wisconsin, Madison, Wisconsin, October 17, 1990; Symposium on Computer Architecture on Professor Zuse’s Birthday, GMD, Schlob Birlinghoven, Federal Republic of Germany, November 30, 1990; New York University, New York City, New York, December 7, 1990; IEEE Seminar, Cambridge, Massachusetts, January 24, 1991.

Arvind. “*T: A Killer Micro for a Brave New World,” MIT-Motorola Meeting, Phoenix, Arizona, January 7, 1991.

Arvind. “*T Multiprocessors,” MIT-Motorola Meeting, Phoenix, Arizona, January 17, 1991; DARPA-MIT Meeting, Roslynn, Virginia, February 20, 1991.

Arvind. “Dataflow Multiprocessors: General Purpose Parallel Machines,” DARPA visit to MIT, January 28, 1991.

Arvind. “The Benefits of Fine-grain Parallelism,” Distinguished Lecturer’s Series, University of Maryland, College Park, Maryland, February 25, 1991.

Arvind. “Implicit Parallel Programming and Dataflow Architectures,” DARPA/ISTO Principal Investigator’s Meeting, Providence, Rhode Island, February 28, 1991; Functional Programming Languages and Computer Architecture (FPCA) Conference, Working Group 2.8, Paris, France, April 18, 1991; Ninth Army Conference in Applied Mathematics and Computing, University of Minnesota, Minneapolis, Minnesota, June 20, 1991.

Arvind. “M-Structures: Parallel Programming with State,” Functional Programming Languages and Computer Architecture (FPCA) Conference, Working Group 2.8, Paris, France, April 19, 1991.

Arvind. “A Syntactic Approach to Program Transformations,” ACM SIGPLAN & IFIP Symposium on Partial Evaluation and Semantics-Based Program Manipulation 1991, Yale University, New Haven, Connecticut, June 17-19, 1991.

Brobst, S.A. “The Evolution of Dataflow Architectures from Static Dataflow to P-RISC,” Monash University, Clayton, Australia, August 13, 1990; Swinburne Institute of Technology, Hawthorn, Australia, August 15, 1990; Boston University, Boston, MA, April 16, 1991.

Nikhil, R.S. "The Parallel Programming Language Id," Visit by Dr. George Cotter, Chief Scientist, National Security Agency at MIT LCS, July 23, 1990

Nikhil, R.S. "Compiling Id for Stock Hardware, using P-RISC," 3rd Workshop on Programming Languages and Compilers for Parallel Computing, Irvine, California, August 1-3, 1990.

Nikhil, R.S. "Compiling a Functional Language for Fine-grained Parallelism," Dagstuhl Seminar on Functional Languages: Optimization for Parallelism, Schloss Dagstuhl, West Germany, September 4, 1990.

Nikhil, R.S. "A Language and Compiler for Fine-grained Threads," University of Wisconsin, Madison, Wisconsin, September 26, 1990; Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Illinois, September 27, 1990; NEC Research Institute, Princeton, New Jersey, October 5, 1990.

Nikhil, R.S. "Massive, Fine-grained parallelism: The P-RISC approach," Thinking Machines Co., Cambridge, Massachusetts, December 18, 1990.

Nikhil, R.S. "Parallel Compilation using P-RISC Graphs," DEC Cambridge Research Laboratory, Cambridge, Massachusetts, February 28, 1991; IBM Almaden Research Center, San Jose, California, March 5, 1991; Dept. of Information and Computer Science, University of California, Irvine, California, March 7, 1991; ATT Bell Laboratories, Murray Hill, New Jersey, March 11, 1991; Second OSU Workshop on Parallel Computing, Dept. of Computer and Information Science, Ohio State University, Columbus, Ohio, March 27, 1991; Computer Architecture Section, Electrotechnical Laboratories, Tsukuba, Japan, April 2, 1991; Computer System Research Laboratory, C&C Research Laboratory, NEC Kawasaki, Japan, April 4, 1991; Institute for New Generation Computing (ICOT), Tokyo, Japan, April 5, 1991; Osaka University, Osaka, Japan, April 8, 1991; Kansai Laboratory, Oki Electric Industry, Osaka, Japan, April 9, 1991; Annual Meeting of IFIPS Working Group 2.8 on Functional Programming Languages, Paris, France, April 15, 1991; Paris Research Laboratory, Digital Equipment Corporation, Paris, France, April 22, 1991.

Nikhil, R.S. "General Purpose Parallel Programming Using the Programming Language Id," IBM Almaden Research Center, San Jose, California, May 10, 1991.

Papadopoulos. "Rescheduling 88120 Instruction Streams," Motorola Microprocessor Products Group, Austin, Texas, August 23, 1990.

Papadopoulos. "Monsoon: A Multiprocessor Architecture Suitable for Intelligent Control," IEEE Symposium on Intelligent Control, Philadelphia, Pennsylvania, September 6, 1990.

Papadopoulos. "Heterogeneous Supercomputers," DARPA Contractor's Meeting, Chapel Hill, North Carolina, October 3, 1990.

Papadopoulos. "The Quest for the Right Multiprocessor Building Block," Microprocessor Forum, San Francisco, California, October 11, 1991.

Papadopoulos. "Virtualized Multithreading: The Right Multiprocessor Building Block?" Thinking Machines Corporation, Cambridge, Massachusetts, February 5, 1991.

Papadopoulos. "Trends for the 90's: Multiprocessing, Multimedia and Multi-networks," Forum on Informatics, American-Hellenic Chamber of Commerce, Athens, Greece, March 28, 1991.

Papadopoulos. "All Parallel Machines Will Look the Same," 17th Annual Asilomar Microcomputer Workshop, April 24, 1991.

Papadopoulos. "*T: 88110-Based Building Blocks for General Purpose Multiprocessors," Motorola Microprocessor Group, Austin, Texas, May 24, 1991.

References

[1] Z. M. Ariola and Arvind. Compilation of Id$^-$: a subset of Id. Technical Report CSG Memo 315, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA, July 1990. Revised 1 November.

[2] Z. M. Ariola and Arvind. A Syntactic Approach to Program Transformations. In *Proc. ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation, Yale University*, June 1991. Also: CSG Memo 295, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA.

[3] Arvind, S. K. Heller, and R. S. Nikhil. Programming Generality and Parallel Computers. *Fourth International Symposium on Biological and Artificial Intelligence Systems*, Sept. 1988.

[4] Arvind and R. A. Iannucci. Two Fundamental Issues in Multiprocessing. In *Proceedings of DFVLR - Conference 1987 on Parallel Processing in Science and Engineering, Bonn-Bad Godesberg, W. Germany, Springer-Verlag LNCS 295*, June 25-29 1987.

[5] M. Beckerle. Generating MINT: The Application of MUD to Monsoon. Technical report, Motorola MCRC, 1 Kendall Square, Building 200, Cambridge, MA 02139, USA, October 3 1990.

[6] M. Beckerle and J. Young. Monsoon Instruction Subsets. Technical report, Motorola MCRC, 1 Kendall Square, Building 200, Cambridge, MA 02139, USA, September 27 1990.

[7] S. Brobst. Operator Definitions for Storage Management in MINT. Technical report, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA, January 8 1991.

[8] P. J. Burns, M. Christon, R. Schweitzer, O. M. Lubeck, H. J. Wasserman, M. L. Simmons, and D. V. Pryor. Vectorization of Monte Carlo Particle Transport: An Architectural Study Using the LANL Benchmark "Gamteb". In *Proceedings Supercomputing '89*, pages 10–20. IEEE Computer Society and ACM SIGARCH, Nov. 1989.

[9] D. Choi. Optimizing Array Bound Checking in a Dataflow Compiler. Bachelor's Thesis, May 1991.

[10] W. D. Clinger. How to Read Floating Point Numbers Accurately. In *SIGPLAN '90 Conference on Programming Language Design and Implementation*, pages 92–100, White Plains, NY, June 1990. ACM.

[11] W. P. Crowley, C. P. Hendrickson, and T. E. Rudy. The SIMPLE Code. UCID 17715, Lawrence Livermore Laboratory, Feb. 1978.

[12] D. E. Culler, A. Sah, K. E. Schauser, T. von Eicken, and J. Wawrzynek. Fine-grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract

Machine. In *4th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, April 1991.

[13] P. Hudak and P. Wadler (editors). Report on the Programming Language Haskell, A Non-strict Purely Functional Language (Version 1.0). Technical Report YALEU/DCS/RR777, Yale University, Department of Computer Science, Apr. 1990.

[14] J.-J. Lévy. An algebraic interpretation of the $\lambda\beta$K-calculus and a labelled λ-calculus. In *λ-Calculus and Computer Schience Theory, Italy (Springer-Verlag Lecture Notes in Computer Science 37)*, March 1975.

[15] R. S. Nikhil. *Id Version 90.0 Reference Manual.* CSG Memo 284-1, July 1990.

[16] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *T: a Killer Micro for a Brave New World. Technical Report CSG Memo 325, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA, January 5 1991.

[17] J. Steele, Guy L. and J. L. White. How to Print Floating-Point Numbers Accurately. In *SIGPLAN '90 Conference on Programming Language Design and Implementation*, pages 112–126, White Plains, NY, June 1990. ACM.

[18] K. Traub. MINT White Paper. Technical Report MCRC-TR-2, Motorola MCRC, 1 Kendall Square, Building 200, Cambridge, MA 02139, USA, October 31 1989.

[19] K. R. Traub. A Compiler for the MIT Tagged-token Dataflow Architecture. Master's thesis, MIT, Aug. 1986.

[20] L. G. Valiant. A Bridging Model for Parallel Computation. *Comm. of the ACM*, 33(8):103–111, August 1990.

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

Contents