International Conference on
Computer System and Education, IISc, 1994

*T (StarT) the Next Generation: In the Real World *

Arvind, Derek Chiou, and Boon Seong Ang
Laboratory for Computer Science
M.I.T

Abstract

The cost of developing state-of-the-art hardware for parallel machines is enormous, and has not been justified by commercial sales. Most researchers have reached the conclusion that parallel machines must ride the technology curve of microprocessors and software used in workstations to keep development costs manageable. In this real world one needs to find a way around the deficiencies of commercial microprocessors from the point of view of using them in parallel machines. The *T project, an MIT-Motorola collaboration, is about providing support for global cache-coherent memory and fine-grain message passing using stock microprocessors. We will discuss our experience of building the hardware and software for the Monsoon dataflow machine, and its influence on the new *T architecture. We will also discuss to what extent *T can support fine-grain parallelism and implicitly parallel programming. Please note that this paper is essentially a summary of [2].

*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-92-J-1310.

1 Introduction

Widespread use of parallel machines in the future is indisputable, yet parallel machines have had little commercial success so far. Some people have attributed the lack of commercial success of parallel computing to rapid improvements in the performance of microprocessors in the last decade. In our opinion the dominant reason is the lack of software. Applications cannot be ported to parallel machines without tremendous effort. Furthermore the effort is usually architecture specific. It is commonly recognized that popular sequential languages such as C and Fortran need to be extended to express parallel computation, since state-of-the-art compiling techniques are unable to extract satisfactory performance from standard languages for the current generation of parallel architectures. There is also some consensus about emerging data parallel languages such as HPF (High Performance Fortran). It is well known, however, that data parallel model of computation is not suitable for irregular parallel computations.

What should be the features of a model to support general-purpose parallel computation? What assumptions should be made regarding the architecture of parallel machines to develop such a model? A lively debate goes on regarding the extent to which the features of parallel architectures should be hidden from the applications programmers. Dataflow reseachers over the years have taken an extreme view in this spectrum. We have argued that declarative programming, because of its implicit parallelism, offers a solid basis for programming parallel machines. Unlike squential languages, declarative languages do not obscure the inherent parallelism of an algorithm. In addition to finding parallelism, however, the difficulty of achieving good speedups has a lot to do with the underlying architecture of parallel machines. Unless a machine supports global memory accesses and synchronization efficiently, compilers have a very difficult task in code generation even from a highly parallel langauge.

In Section 2 we briefly describe our experience in building the Monsoon dataflow machine and programming it in Id[7], an implicitly parallel language. Next we descibe the *T architecture, which was motivated by our desire to bring dataflow ideas closer to commercial parallel machines. We then describe an implementation of the *T architecture using modified Motorola 88110 processors. Finally, we descibe the latest version of the *T architecture (called *T-NG), which integrates globally coherent caches with efficient message passing.

2 The Monsoon Project

The Monsoon dataflow processor[3, 9] (see Figure 1), a 10 MHz full custom processor designed at MIT and built by Motorola, incorporates most of the features deemed necessary to run implicitly parallel programs. It has very fast synchronization mechanisms, fast context switch abilities and an excellent

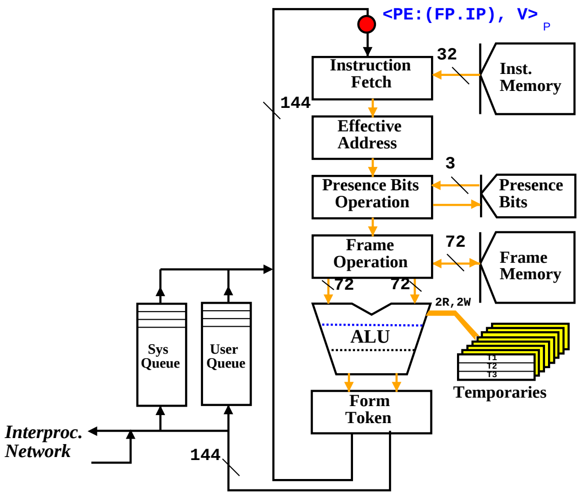

Figure 1: A Monsoon node (taken from [3])

Program	$\frac{1PE}{1PE}$	$\frac{1PE}{2PE}$	$\frac{1PE}{4PE}$	$\frac{1PE}{8PE}$
4 × 4 Blocked-MM 500 × 500	1	1.99	3.90	7.74
Gamteb-2c 40000 particles	1	1.95	3.81	7.35
Simple 100 iters, 100 × 100	1	1.86	3.45	6.27
Paraffins n = 22	1	1.99	3.92	7.25

Figure 2: Speedup Results: Id on Monsoon (taken from [4])

network interface to a very fast network (100 MBytes/sec/port)[5]. Two machines, each containing eight processors and eight synchronizing memory boards, along with numerous single processor single synchronizing memory board units were built. One of the large machines is at MIT while the other is at Los Alamos National Laboratory.

Due to lack of manpower, the only compiler available for Monsoon is an Id compiler. We ran numerous studies on Monsoon to assess its performance characteristics[4]. Some of our performance results are shown in Figure 2 and Figure 3. Figure 2 gives speedup results for the Monsoon processor, from 1 processor to 8 processors. Monsoon's speedup numbers are generally excellent.

Figure 3 compares instruction counts of Monsoon running Id to a MIPS R3000 running optimized C or Fortran codes. The basic conclusion from the second table is that the in-

Program	25 MHz R3000		10 MHz Monsoon	
	$\times 10^6$	secs	$\times 10^6$	secs
Matrix-Multiply, 500 $\times$ 500				
double precision	1198	202.3	1768	176.8
single precision	915	153.1	–	–
Blocked-MM, 500 $\times$ 500				
double precision	954	61.4	1058	105.8
single precision	741	44.9	–	–
Gamteb-2c 40000 particles	265	11.1	590	59.0
Simple, 100 iters, 100 $\times$ 100				
double precision	1787	86.5	4682	468.2
single precision	1745	84.1	–	–
Paraffins n = 22	102	12.0	322	32.2

Figure 3: Performance of Fortran/C on the MIPS R3000 verses Id on 1 processor Monsoon (taken from [4])

struction count for a MIPS processor is not more than 3 times lower than for Monsoon. Much of this performance difference is due to the fact that our source language, Id, is very high level and is thus hard to compile efficiently. In addition to the source language, however, we believe that this number reflects a number of deficiencies in Monsoon including poor single-threaded performance, a weak instruction set with poor memory addressing abilities and very few registers for intermediate values, and hardware optimized for synchronization which essentially forces synchronization to be used more than is actually necessary.

Though these architectural weaknesses could be corrected with another implementation, we felt that it was not worth the effort to do so. Monsoon is a full custom processor and thus cannot leverage any commercial hardware or software. Without leveraging commercial products, there is little chance even for a well-endowed research group to make a performance-competitive machine. It is difficult to justify building the hardware and software for an experimental machine, if commerical machines out-perform it. For these reasons, we felt that we must leverage commercial products in order to demonstrate that the ideas we propose have merit outside of academia. The next few sections discuss how we propose to do so.

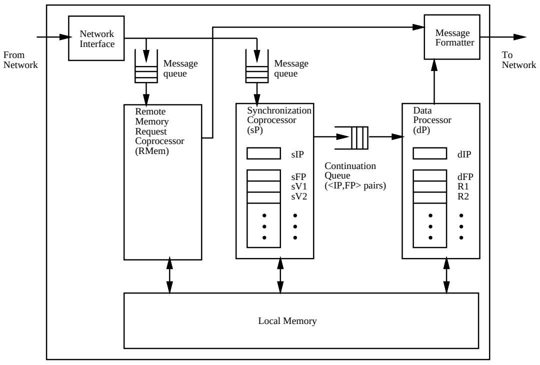

Figure 4: A *T node (taken from [8])

3 The *T Architecture

The *T architecture (pronounced Star T or start) was an attempt to build a processor that would be a good general-purpose building block for a parallel machine. It is based on an augmented RISC pipeline core. As shown in Figure 4, *T consists of three parts: a data processor, a synchronization processor and a remote memory processor. The data processor is essentially the same as a conventional RISC processor with the exception that it has the ability to get work from the synchronization processor and that it is able to send messages to the network. The synchronization processor synchronizes messages from the network and is able to write parts of a network message into memory that the data processor can read. Once a synchronization event is complete, a continuation is enqueued for the data processor. The remote memory processor services memory requests from other processors and returns the results via the network. The synchronization processor and the remote memory processor can be thought of as coprocessors that help process messages from the network. Since modern RISC pipelines are optimized for long threads of execution, the coprocessors are inserted to reduce the need to interrupt the data processor, allowing it to execute longer threads than would be possible otherwise.

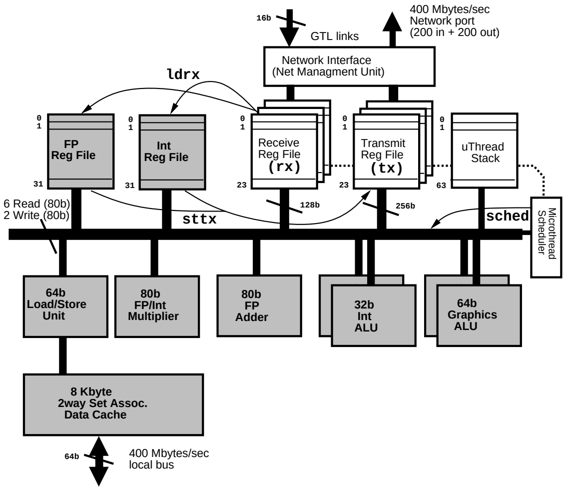

Figure 5: The 88110MP. (taken from [10])

*T was designed with a practical implementation in mind. Though the coprocessors are conceptially separate, it is possible to fold the functionality of the synchronization processor and remote memory processor into a single pipeline; other combinations are feasible as well. Thus a *T node could be produced with slight modifications to a conventional processor. This was the basis for the 88110MP *T project described next.

4 Motorola 88110MP Implementation of *T

The 88110MP *T is based on an augmented Motorola 88110 processor (see Figure 5). The modifications provided an excellent network interface along with support for microthreads. Since the network interface is put on-chip, it is accessed via register move operations, making it especially fast. The 88110MP *T leverages the effort Motorola put into the 88100 architecture while providing much of the support necessary for running our programs. The 88110 architecture is particularly suited for the modifications we desired, allowing us to move up to 256 bits of data to the network every cycle.

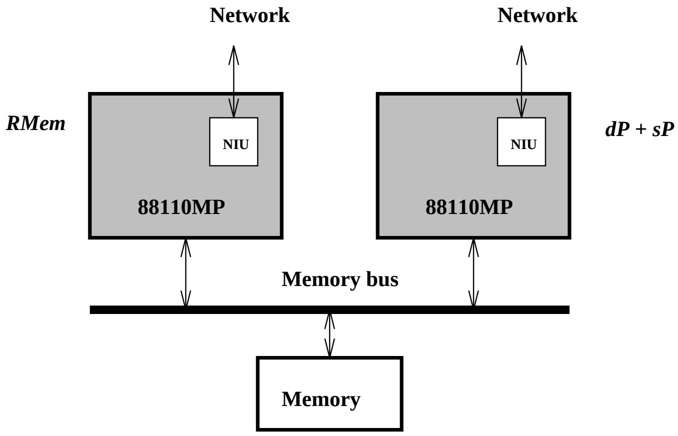

Figure 6: An 88110MP *T node

Two 88110MP's on a snoopy bus and sharing some memory would make up a *T *site* (see Figure 6). We felt that one of the 88110MP's would emulate the remote memory processor and the other 88110MP would act as the data and synchronization processors. Thus, using a single building block, we would be able to implement *T functionality.

Though the modifications were not especially difficult, they did require full custom design and, more importantly, additional pins on the package. Even these small changes can dramatically increase the development costs, since new boards must be developed for the processor and new masks have to be designed and produced for the added functionality.

Because of Motorola's shift in emphasis from the 88100 architecture to the PowerPC architecture and given the cost of completing the 88110MP implementation, it was decided that we change processors. Our next project, *T the Next Generation (*T-NG) is based on stock PowerPC 620 microcprocessors. That project is discussed in the next section.

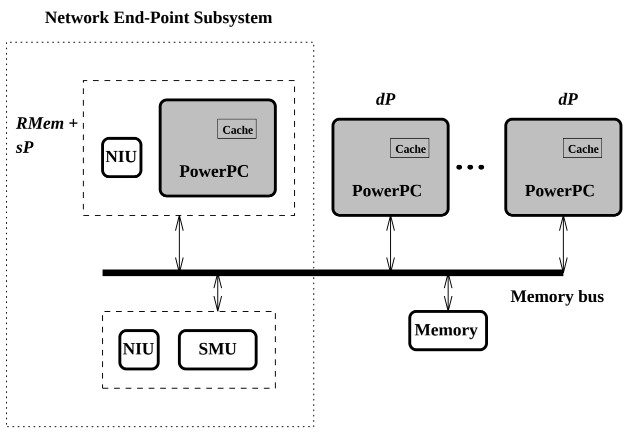

Figure 7: A *T-NG site

5 *T The Next Generation

Unlike the 88110MP, the PowerPC-based *T-NG provides an external network interface unit (NIU) and shared memory unit (SMU). Both the NIU and SMU work in conjunction with another dedicated PowerPC to provide message passing and cache coherency support. We expect to ride the technology curve by not modifying the microprocessor. We can also expect to leverage operating system and basic compiler technology from the manufacturer, saving us a huge amount of work.

Since current generation conventional microprocessors, including the 620, support snoopy bus based cache coherency protocols in hardware, we decided that support for caching global memory would be inexpensive and could provide a very efficient communication mechanism. Snoopy bus coherency protocols are easy to extend to a general global coherency protocol, requiring very little additional hardware. Simple accesses to global data, therefore, are performed by a single instruction. When there is a cache hit, the desired value is available almost immediately afterwards. Only when there is a cache miss does network access occur, and this access is performed by hardware, eliminating overhead for the processor. Thus, it is quite possible that on *T-NG global caching will provide a faster communication mechanism is many cases than message passing, especially since there is no hardware support within the processors for message passing.

A *T-NG site will consist of some number (probably 4 or less) of 620 processors connected on a snoopy bus (see Figure 7). A network end-point sub-system, consisting of another 620 with an NIU and an SMU will be connected onto the snoopy bus as well. Messages sent to the network from the data processors will be written to a shared software queue that the network end-point can read. Actual transfer of the message into the network is performed by the network end-point subsystem. Likewise, the network end-point will write messages received from the network into a shared queue that the receiving processor can read. Global cached shared memory will be supported by the SMU which will also sit on the snoopy bus. The SMU monitors memory operations on the bus, initiating remote memory operations to complete operations on global memory that miss in the cache. The SMU will format, send and receive messages dealing with global memory accesses that travel through the cache. It is likely that the network end-point sub-system will handle most of the coherence protocol processing at the home site, using the end-point sub-system's 620 to provide the processing power that is required to implement these protocols.

*T-NG clearly incorporates many concessions in the name of practicality. Since a site will consist of a stock Motorola multiprocessor board with a custom network end-point subsystem attached, we cannot attach network interfaces to each of the processors in the machine (such a configuration is shown in Figure 8). Reliance on as much stock hardware as possible, however, insures that the system will be cost competitive with comparable commerical systems, and that most

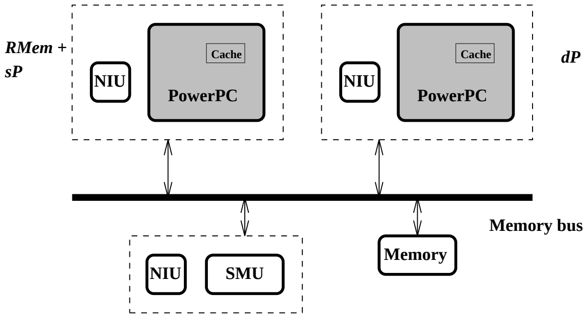

Figure 8: An alternative *T-NG site

if not all of the software written for the stock hardware can be reused by our machine. These advantages convinced us to go forward with this configuration.

Currently, *T-NG is still in the design phase — the network end-point subsystem is being worked out. We hope to have the high level design completed by the middle of 1994 so that Motorola can start the implementation.

6 Conclusion

In this paper we have discussed the lessons we learned from the Monsoon project. Monsoon was a fully custom, pure dataflow machine. Though it was very successful running fine-grain parallel code, for pragmatic reasons we were forced to move to machines built from stock processors. We also briefly presented the *T model which was motivated by the desire to create a better building block for parallel machines. We then discussed our first implementation of a *T node which was based on a Motorola 88110 RISC processor. This project was abandoned in favor of a move towards the PowerPC architecture family. We took the opportunity to incorporate global caching into the *T model. The resulting model is called *T-NG for *T, the Next Generation.

*T-NG is similar to many existing machines. Japan's Electrotechnical Laboratory's (ETL) EM-4[11] and its descendant architectures are custom machines designed to execute multithreaded code. They have imbedded network interfaces which allow messages to be sent in a single cycle and hardware support for synchronization as well. *T-NG is very similar to MIT's Alewife[1] and Stanford's FLASH[6]. Both machines support global cached memory as well as message passing, and both are based on stock processors. It is exciting to see that parallel architectures are converging. The machines mentioned differ only in detail from *T-NG. Eventual success of these attempts will depend very much on the quality of the hardware implementations, and the system software that is provided for using these machines.

7 Acknowledgements

The discussion on *T-NG has involved many people, both within our group and at Motorola. We have had very extensive discussions with Mike Beckerle, Bob Greiner and Jamey Hicks at Motorola and with Greg Papadopoulos, James Hoe, John Morris, S.K. Nandy, Chris Joerg, and Andy Boughton at MIT.

References

[1] A. Agarwal, D. Chaiken, G. D'Souza, K. Johnson, D. Kranz, J. Kubiatowicz, K. Kurihara, B. H. Lim, G. Maa, D. Jussbaum, M. Parkin, and D. Yeung. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. In *Proceedings of Workshop on Scalable Shared Memory Multiprocesors*. Kluwer Academic Publishers, 1991.

[2] Arvind, B. S. Ang, and D. Chiou. StarT the Next Generation: Integrating Global Caches and Dataflow Architecture. In *Proceedings of the ISCA '92*

Dataflow Workshop, Hamilton Island, Australia (to be published), May 1992.

[3] D. E. Culler and G. M. Papadopoulos. The Explicit Token Store. *Journal of Parallel and Distributed Computing*, 10(4):289–308, 1990.

[4] J. Hicks, D. Chiou, B. S. Ang, and Arvind. Performance Studies of the Monsoon Dataflow Processor. *Journal of Parallel and Distributed Computing*, 18(3):273 – 300, 1993.

[5] C. Joerg and G. A. Boughton. The Monsoon Interconnection Network. In *Proceedings of the 1991 IEEE International Conference on Computer Design, Cambridge, MA*, Oct 1991.

[6] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Stanford FLASH Multiprocessor. In *Proceedings of The 21th Annual International Symposium on Computer Architecture, Chicago, Il*, 1994.

[7] R. S. Nikhil. Id Reference Manual, Version 90.1. CSG Memo 284-2, Laboratory for Computer Science, MIT, Cambridge MA, Sep 1990.

[8] R. S. Nikhil, G. M. Papdopoulos, and Arvind. *T: A Multithreaded Massively Parallel Architecture. In *Proceedings of 19th Annual International Symposium on Computer Architecture, Gold Coast, Australia*, May 1992.

[9] G. M. Papadopoulos. *Implementation of a General-Purpose Dataflow Multiprocessor.* MIT Press, 1992.

[10] G. M. Papadopoulos, G. A. Boughton, R. Greiner, and M. J. Beckerle. $\star$T: Integrated Building Blocks for Parallel Computing. In *Proceedings of Supercomputing '93, Portland, OR*, Nov 1993.

[11] S. Sakai, Y. Yamaguchi, K. Hiraki, Y. Kodama, and T. Yuba. An Architecture of a Dataflow Single Chip Processor. *Proceedings of the 16th Annual International Symposium on Computer Architecture, Jerusalem, Israel*, pages 46–53, May 1989.