RETROSPECTIVE:

The J-Machine

William J. Dally[1], Andrew Chang[1], Andrew Chien[2], Stuart Fiske[9], Waldemar Horwat[4], John Keen[9], Richard Lethin[5], Michael Noakes, Peter Nuth[6], Ellen Spertus[7], Deborah Wallach[8], D. Scott Wills[3]

[1] Computer Systems Laboratory, Stanford University

[2] Department of Computer Science, University of Illinois, Urbana-Champaign

[3] Department of Electrical and Computer Engineering, Georgia Institute of Technology

[4] Netscape Communications

[5] Equator Technologies Consulting

[6] Hewlett Packard Laboratories

[7] Department of Computer Science, Mills College

[8] DEC, Western Research Laboratory

[9] Silicon Graphics Computer Systems

Eleven years ago, at ISCA 14, we published a paper titled, "Architecture of a Message-Driven Processor" [1] marking the start of our J-Machine project at MIT. The project culminated with the construction of a working prototype in 1991 [2] and the evaluation of this prototype in 1992 [12, 15].

The J-Machine demonstrated the use of a *jelly-bean* part, a commodity part incorporating a processor, memory, and a fast communication interface, as a building block for computing systems. It was a *fine-grain* parallel computer designed to exploit large amounts of parallelism by balancing the use of silicon area between processor and memory. The J-Machine provided a small set of efficient communication and synchronization *mechanisms* that were used to support a broad range of programming models. It also provided fast user-to-user messaging without software intervention by having each message dispatch a message handler.

This retrospective reviews the history of the J-Machine project, discusses its contributions with the perspective of hindsight, and assesses what was learned from the project

Chronology

The chronology of the J-Machine project is summarized in Table 1. The project started at MIT in 1986. A team of students built a simulator for the machine, and we published a concept paper at ISCA in 1987. Funding to build the machine was secured in the summer of 1988. Intel joined us as an industrial partner and implementation started that August. We designed the machine using a very effective home-grown set of CAD tools [7] for

Table 1 Project Chronology

Date	Description
Sept 86	Project start
May 87	Initial architecture studies reported
Aug 88	Implementation started
Dec 90	Layout complete
Jun 91	First silicon
Jan 92	Second pass silicon
May 93	Evaluation reported

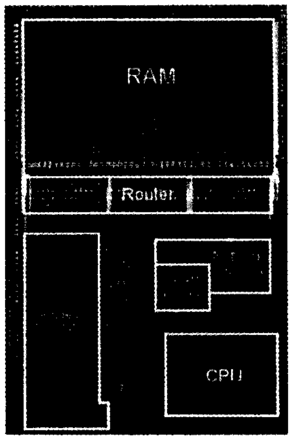

Figure 1: MDP Chip

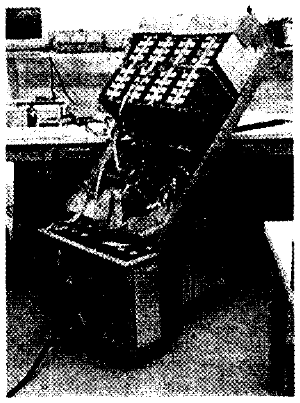
Figure 2: 1024-Node J-Machine

logic design, and Intel tools for physical design and verification. Layout was completed in December of 1990. The first chips arrived in June of 1991 and were running programs within a few hours. After revising the chips in early 1992 to correct a few bugs, we built three J-Machines: a 1024-node machine at MIT and 512-node machines at Caltech and Argonne. In parallel with the hardware development, several software systems were built for the machine [4, 19, 14, 9].

A die photo of the MDP chip with outlines identifying the major components is shown in Figure 1. The chip measures 1 x 1.5cm in a 1.2µm CMOS process. It was designed primarily using standard cells with hand placement for the data paths. The 32-bit integer CPU, at the lower right, measures 3.7 x 2.9mm.

Figure 2 shows a 1024-node J-Machine with the skins removed. Sixteen processor boards containing 64-nodes each occupy the top portion of the machine. The board stack communicates through elastomeric connectors [10]. The base of the machine contains an array of up to 80 disks. The peripheral bay is just below the processor stack. Peripheral interface cards developed for the machine include disk controllers, a distributed frame buffer for graphics, and two host interfaces.

Contributions

1. The Jellybean Part

The MDP chip was a prototype of our vision of a *jellybean* or commodity part: adding a processor and network hardware to a commodity memory chip. In the original paper we cited the advantages of high bandwidth, low-latency access to the on-chip memory and pointed out that the low latency access to the memory of other nodes in the network prevented the small amount of memory per node from limiting applications. The amount of memory reachable in a given number of cycles is important, not the amount of memory per node.

Our original plan was to build our own DRAM memory for the MDP prototype. Implementation constraints drove us to implement the actual MDP with 144Kbits of on-chip SRAM and 8Mbits of off-chip DRAM. This configuration simulated what we expected could be placed on-chip in the next generation (16Mbit) DRAM technology.

After demonstration of the prototype J-Machine in 1991, one of us (Dally) visited all of the major DRAM manufacturers to propose joint development of a commercial jellybean part by augmenting a 16Mbit DRAM with a 32-bit RISC processor, network interface, and router. There was no industrial interest in the project at the time.

Recently the idea of building jellybean parts combining a processor and memory, but without the network interface, has been revived [5, 13]. These projects are aimed at building stand-alone single-node systems, however, and do not address our original goal of developing building blocks for fine-grained parallel computer systems.

2. Fine-Grain Parallel Computing

Fine-grain machines, that balance processor and memory by silicon area rather than MIPS/Mbyte, achieve significantly better throughput per unit area, and per dollar, than do conventional coarse-grain machines [3]. This efficiency is achieved by having a larger fraction of *working* silicon and by reusing expensive memory more often. Efficient communication and synchronization mechanisms are needed to realize the potential of fine-grain machines, or these gains are swamped by overhead.

The bulk of the real cost (silicon area) in conventional computer systems is in memory. One can build a very competent 32-bit RISC processor (not including cache) in the area required by 100Kbytes of DRAM. Adding pipelined floating-point arithmetic raises this number to 500Kbytes. A machine with a simple pipelined floating-point processor and 256Mbytes of memory has only 0.2% of its silicon devoted to processing. This fraction of *working* silicon is decreasing with time as the memory in machines balanced by MIPS/Mbytes increases.

Conventional systems raise the fraction of working silicon by devoting large amounts of silicon to a single processor in an attempt to exploit instruction-level parallelism. This gives rapidly diminishing returns in performance per unit area. Doubling the area of a simple pipelined processor, for example, typically yields a performance improvement of less than 30%. The gains from a second doubling are significantly smaller. Increasing the number of processors, which gives nearly linear returns for some demanding programs, is far more efficient.

Consider an MDP built with today's technology incorporating a simple pipelined floating-point processor with a 64-Mbit DRAM. The fraction of working silicon on each of these modern jellybean chips would be 6%. A fine-grain machine built from 32 of these chips would have the same memory capacity and would cost about the same (same silicon area) as a 256Mbyte workstation with a more aggressive superscalar processor. The fine-grain machine has much higher memory bandwidth, lower local memory latency, and much greater peak performance. Even if our simple processor runs serial applications at half the speed of the more aggressive workstation processor, it outperforms the conventional processor by a factor of 16 on parallel applications and needs an average parallelism of only two to break even.

The efficiency of fine-grain computers depends on the availability of parallelism in applications. Our application studies on the J-Machine showed that there is plentiful parallelism in many applications. At small problem sizes, however, exploiting this parallelism requires short threads and frequent inter-thread interaction. The MDP's fast mechanisms enabled us to extract large amounts of parallelism even from applications run on small problem sizes. One string manipulation application, for example, gave a speedup of over 200 running a 1024-character string on 512 processors [12].

3. Mechanisms vs. Models

The J-Machine was designed with a set of fast primitive mechanisms for communication and synchronization intended to support a broad range of programming models. A message could be sent from user level with a single instruction, a message handler was dispatched by hardware on message arrival, synchronization was supported with presence bits on all memory locations and registers, and global (segmented) addressing was supported across the machine. These mechanisms were used to implement object-oriented [4], data flow [14], and message-passing [9] programming systems on the prototype hardware.

In 1987 when the MDP was proposed, people built machines specialized for one model of computation: data flow machines, shared memory machines, message-passing machines, and so on. It was generally accepted that a hardwired implementation of the programming model was required to achieve good performance. The J-Machine demonstrated that this was not the case.

The idea of building mechanisms in hardware and implementing programming models in software has received considerable attention [11][6]. However, two trends threaten this approach to parallel machine design. On the hardware side, the deep and complex on-chip memory hierarchy of modern microprocessors makes it difficult to build mechanisms without building a custom processor. The DEC Alpha 21164 processor used on the Cray T3E, for example, takes at least 20 cycles to wiggle a pin of the chip in response to a store instruction. It would not be difficult to add a path that bypasses the memory hierarchy and gives a much faster response.

On the software side, there is a disturbing trend toward applications that are written for the *least-common denominator* machine. Message-passing application are written in MPI or PVM and tuned to run on machines with 100μs message latency. Shared memory applications are written using a static process structure and tuned to operate with 10μs synchronization times. It is no wonder the people who write these applications conclude that they need large problem sizes to extract parallelism. Unfortunately, programs that are tuned to run on machines with slow mechanisms can't take advantage of fast mechanisms when they are available.

4. Fast Message Handling

In 1991, the J-Machine had the fastest interconnection network of any parallel machine in terms of start-up latency, throughput, and latency per hop [12]. Its performance was not surpassed until the announcement of the Cray T3D in October of 1993. The J-Machine's communication performance was due to a combination of a fast router, 3D packaging, and a streamlined network interface. Many ideas from the J-Machine network fabric have found their way into commercial machines including the Cray T3D and T3E and the Intel ASCI Red machine.

The network interface of the MDP, by providing a SEND instruction to transmit messages and hardware message dispatch on reception, set the standard for user-level messaging performance. Other experimental machines have taken a similar approach [11, 6, 17]. Software protocols have also evolved [16] to provide the same model of reactive messaging on stock hardware.

Lessons Learned

Fine-grain computing is feasible: Our experiments showed that it is feasible to extract parallelism with a thread size of a few hundred instructions from many applications and that with efficient mechanisms these applications can be efficiently executed on a machine with just 1Mbyte of memory per node.

Mechanisms work: We implemented several parallel programming models with performance competitive with hardwired implementations.

Building programming systems using the J-Machine mechanisms also taught us their shortcomings. For example, the streaming message SEND instruction of the J-Machine causes resource allocation problems. These shortcomings have been remedied in our follow-on projects [18].

There is no substitute to building it: By implementing the MDP we found that our initial estimates of cost and performance were in some cases far from the mark. The building process also fleshed out many challenging design and technology problems. Our work on high-speed signaling, for example, started from an observation that MDP performance was limited by pin bandwidth. The physical 1024-node prototype (too large to simulate) revealed challenging resource allocation and load balancing problems [8]. While it is certainly easier to quit after the simulation stage of a project, we found that the results at that stage lack reality and are often wrong.

Focus on the core issues: For the MDP the core issues were communication and synchronization mechanisms, fast context switching, and fine-grain thread and object handling. We were not studying instruction set design, pipelining techniques, circuit design, or logic design. We took a very conservative approach to these areas that sacrificed considerable absolute performance but greatly increased our probability of success. At various points in time we considered building our own DRAM, making extensive use of asynchronous logic, and pipelining the processor. In retrospect we are glad that we avoided this temptation of creeping featurism.

Why did we succeed? Intel's solid commitment as an industrial partner provided us with advanced CAD tools, manufacturing, and engineering talent. Their interest and the interest of students was held by a bold research agenda, rather than incremental measurement. The team expended as much time on validation as on design, meticulously cross checking the models at the instruction, RTL, gate and switch level; writing a comprehensive test suite; and simulating heavily for slow paths, hazards, and race conditions. We compromised some dimensions early, such as using standard-cell rather than full-custom design. Finally, the nature of the machine itself was amenable to building large systems: effort focused on the single-chip building block was multiplied by its ability to scale and form a large parallel machine.

A machine that requires new software is a hard sell: Technology transfer depends more on ease of insertion than on utility. Concepts from the J-Machine network, which are easy to insert, have seen the most use. The fast message interface, synchronization mechanisms, and fine-grain architecture have been largely ignored because they require major modifications to a microprocessor, completely new software, or both. The barrier to new ideas represented by existing software is formidable.

Incidental errors: We made a number of errors in the architecture of the MDP that were orthogonal to the issues of granularity and mechanisms. The machine had inadequate floating-point performance, too few registers, and did not operate the on-chip memory as a cache. These errors hindered our evaluation of the machine and were a barrier to getting applications programmers to target the prototype machine. Also, our software development would have been simplified if we had extended an existing instruction set rather than developing a new one.

Conclusion

The vision of a fine-grain parallel computer constructed from *jellybean* processor-memory-network components is even more compelling today than it was in 1987. As the fraction of working silicon in modern machines and the performance returns from aggressive superscalar implementations continue to decrease, explicitly parallel machines look even more attractive. Unfortunately, software compatibility remains a formidable barrier.

Using a fine-grain parallel computer as the memory of a conventional coarse-grain machine can lower the barrier of software compatibility. Such a *smart memory* machine can run existing programs unchanged, albeit with some increase in cost. Programs can then be parallelized incrementally, modifying critical loops and kernels one at a time to run on the fine-grain computer. By offering an incremental software path, smart memories make possible the transition from coarse-grained machines based on instruction-level parallelism to fine-grained machines based on explicit parallelism.

The J-Machine demonstrates the importance of building experimental computer systems. In the academic world, where we can afford to fail, we can build a machine based on unproven ideas. We also have the luxury of building a machine that demonstrates a vision of a computer system without concern for compatibility. It is essential to build and evaluate such machines. Simulation results do not reduce the perceived risk of new technologies sufficiently for industry to adopt them.

Acknowledgments

We thank all of the many people who contributed to the success of the J-Machine project. Special thanks go to Scott Furman on the MIT side and to Albert Yu, Greg Fyler and Roy Davison who made it all happen on the Intel side. We also thank Mark Pullen, John Toole, and Gil Weigand at ARPA for their vision and support.

References

[1] W. Dally, et al., "Architecture of a Message-Driven Processor," *ISCA-14*, pp. 189-196, 1987.

[2] W. Dally, et al., "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms," *IEEE Micro*, pp. 23-39, April 1992.

[3] W. Dally, "A Universal Parallel Computer Architecture," *New Generation Computing*, pp. 227-249, 1993.

[4] W. Horwat, A. Chien, and W. Dally., "Experience with CST: Programming and Implementation," *PLDI*, pp. 101-109, 1989.

[5] P. Kogge, et al., "Combined DRAM and Logic Chip for Massively Parallel Systems," *Proc. 16th Conf. On Advanced Research in VLSI*, Computer Society Press, pp. 4-16, 1995.

[6] J. Kuskin, et al., "The Stanford FLASH Multiprocessor," *ISCA-21*, pp. 302-313, 1994.

[7] R. Lethin and W. Dally, "MDP Design Tools and Methods," *ICCD*, pp. 424-428, 1992.

[8] Lethin, Richard A., *Message Driven Dynamics*, Ph.D. thesis, MIT, March 1997. Also MIT/LCS/TR-721.

[9] D. Maskit and S. Taylor, "A Message-driven Programming System for Fine-grain Multicomputers," *Software - Practice and Experience.* 24(10), pp 953-980, October 1994.

[10] M. Noakes and W. Dally, "System Design of the J-Machine," *Advanced Research in VLSI*, pp. 179-194, 1990.

[11] R. Nikhil, G. Papadopoulos, and Arvind, "*T: A Multithreaded Massively Parallel Architecture," *ISCA-19*, pp. 156-167, 1992.

[12] M. Noakes, D. Wallach, and W. Dally, "The J-Machine Multicomputer: An Architectural Evaluation," *ISCA-20*, pp. 224-235, 1993.

[13] D. Patterson, et al., "A Case for Intelligent RAM," *IEEE Micro*, pp. 34-44, March/April 1997.

[14] E. Spertus and W. Dally, "Dataflow on a General-Purpose Parallel Computer," *ICPP*, pp. II231-II235, 1991.

[15] E. Spertus et al., "Evaluation of Mechanisms for Fine-Grained Parallel Programs in the J-Machine and the CM-5," *ISCA-20*, 1993.

[16] T. von Eicken et.al, "Active Messages: A Mechanism for Integrated Communication and Computation," *ISCA-19*, pp. 256-266, 1992.

[17] Y.Kodama, et al., "The EM-X Parallel Computer: Architecture and Basic Performance", *ISCA-22*, pp.14-23, 1995.

[18] W. Lee, et al., "Efficient, Protected Message Interface in the MIT M-Machine", *IEEE Computer*, November 1998.

[19] D. Wallach, PHD: *A Hierarchical Cache Coherence Protocol*, S.M. Thesis, MIT, 1992.