Skirting Amdahl's Law: Using SPSD Execution with Optical Interconnects

Doug Burger and James R. Goodman

Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton Street
Madison, Wisconsin 53706 USA

galileo@cs.wisc.edu
http://www.cs.wisc.edu/~galileo

Abstract

Optical interconnects provide new parallel processing opportunities through inexpensive broadcasts high-bandwidth, point-to-point connections. However, the problems of flow control and buffering inhibit current parallel architectures from effectively exploiting the advantages of optical interconnects. We propose using an execution model, called Single Program, Single Data stream (SPSD) to exploit inexpensive optical broadcasts and reduce the serial overheads of parallel programs. We describe one possible implementation of such a system (DataScalar), and discuss how future systems can be designed to better exploit optical interconnects.

1 Introduction

From the architect's perspective, optical interconnects provide a new and interesting set of opportunities for designing multiprocessor systems. Intrinsically high bandwidth, potentially low latencies, and the "free space" nature of optical interconnects all provide new opportunities for both evolutionary and revolutionary changes in multiprocessor architectures.

Optical technologies have yet to move into the mainstream, however, and have so far failed to supplant electrically-based interconnects as the technology of choice for multiprocessor interconnects. This is historically due to a lag in the level of available optical technology compared to the electrical equivalent. More recently, however, cost and utility constraints have eclipsed the optical technology's rapid advances. Evolutionary improvements are insufficient to overcome the long history and tremendous investment in electrical transmission media.

To succeed, optical technologies must demonstrate not just superiority, but a major advance over the dominant technologies they hope to replace. The computer industry has taken its place among the largest industries in the world, while still continuing to produce rapid advances across a wide range of technologies. The semiconductor business is large and well financed, with a history of progress so rapid and consistent that it is hazardous to predict when progress may slow. The massive resources available for the advancement of dominant technologies—benefiting from economies of scale—present an enormous challenge to any new technology, no matter how superior it may appear. Core memory systems continued to survive—and even grow—years after they were predicted to be supplanted with semiconductor RAM technology.

Though optical devices are likely to see increased use in long-haul networks, it is not apparent that they offer a clear benefit to conventional massively parallel machines. These machines already depend on high-bandwidth communication to exploit fine-grained parallelism. They have been well studied, and dramatic improvements are not likely to be achieved by the elimination of a single bottleneck. We will discuss the two major classifications of parallel computers—shared-memory multiprocessors and multicomputers—showing that no single improvement in the interconnect is likely to provide dramatic improvement. Thus optical interconnects are unlikely to become ubiquitous in today's conventional MPPs.

Networks that provide high-bandwidth, low-latency communication between arbitrary nodes are critical to the success of massively parallel computers. Raw bandwidth alone is not sufficient, however. Even if network bandwidth were increased arbitrarily, other bottlenecks would soon emerge, precluding substantial performance gains. In bus-based snooping cache systems, for example, all communications are broadcast, and each such communication requires processing at every node, including a memory

This work was supported in part by NSF Grant CCR-9207971, an unrestricted grant from the Intel Research Council, and equipment donations from Sun Microsystems.

lookup. Thus processing at each node grows linearly with the number of processors in the system.

A major limitation of parallel computing is the challenge of balancing the workload. The availability of twenty processors does not automatically translate into a speedup of twenty, even if the code can be readily parallelized. It may be that some parts take longer to compute than others, or may be unpredictable, so that many of the processors are idle much of the time, waiting for others to finish their assigned work. Systems may have *hot spots* [15] both in processors and in memory, where progress is impeded because a computation, or access to a particular memory location, is required excessively.

Hot spots are particularly troublesome in broadcast networks. A free-space optical interconnection network can achieve extremely high bandwidth because many nodes can be simultaneously broadcasting at a very high rate. Unfortunately, if a single node must respond to a significant portion of the messages being broadcast, it can be rapidly swamped. It must therefore have some mechanism for throttling the sources of the data, a procedure known as *flow control*. This problem emerges as a major limitation of free-space optical interconnections: how to throttle sending nodes that are overloading a particular receiver without interfering with communications among other nodes.

Another limit to the successful implementation of massively parallel systems (hundreds or thousands of processors) is serialized code within applications. Amdahl's Law states that a the speedup of a program is limited by the reciprocal of the serial portion of the program. For example, a program for which 95% of its code can be parallelized will have a speedup of only 20 even if the non-serial portion of the code is performed in zero time. Although optical interconnects may enable very large systems to be constructed, this serial overhead must be considered.

Thus the availability of a high bandwidth, low-latency network (such as might be available using free-space optical interconnect) cannot be much better utilized by conventional MPPs. Architectures that can exploit the high bandwidth capability without having to provide elaborate flow control are the best hope for inclusion of optical interconnect technology. This paper provides an example of a novel architecture that exploits the unique features of optical interconnects to reduce serial overheads within a parallel program. Our assumption is that microprocessors will continue their exponential performance growth, with the consequence that communication becomes relatively much more expensive in future systems. Because computation in such systems will be cheap compared to communication, we propose to perform serial code redundantly at every node, broadcasting all operands needed for that code on the optical interconnect.

We call our proposed execution model Single-Program, Single-Data stream (SPSD), extending Flynn's classification [7]. This execution model was devised for our proposed DataScalar architecture [3], which targets future uniprocessor programs running on a small number of processors. The SPSD execution model was derived from the Massive Memory Machine work of the early 1980s [8]. We believe that the DataScalar concepts are an excellent match for the small fraction of large-scale parallel programs that is difficult or impossible to parallelize. More efficient execution of this serialized fraction will reduce the fraction of execution time that it requires, thus increasing the scalability of the program.

The rest of this paper is organized as follows: in Section 2 we present an overview of multiprocessor architectures, and discuss how optical interconnects may interact with each class of multiprocessors. We also discuss the pitfalls that prevent these architectures from realizing significant benefits from the use of optics. In Section 3 we describe the SPSD execution model, and discuss how it may be applied to massively parallel systems. In Section 4 we engage in speculation as to how optics and parallel architectures may interact in the future. Finally, in Section 5 we provide a summary of our ideas.

2 Interaction of MPP architectures and optics

There are two major categories of multiprocessor architectures that have emerged over the years. In this section, we discuss how each category may or may not benefit significantly from optical (as opposed to electrical) interconnects.

2.1 Shared-memory multiprocessors

The first shared-memory multiprocessors, in which all memory lies equidistant from every processor, were what are commonly called "dancehall" architectures. They were so named because all processors were located on one side of a general interconnect, while all memory modules resided on the other (an eight-processor example is shown in Figure 1). Examples of these machines include the IBM GF-11 [1], the NYU Ultracomputer [10], and the IBM RP3 [17].

This organization had the advantage of a high-bandwidth interconnect with uniform latency to all memory, making the machines easy to program. The drawback to these architectures is the long latency incurred by traversing the multistage interconnect. Placing caches at the processor side of the interconnect can reduce the average memory latency, but introduces the well-known cache coherence problem, which is difficult to solve for such architectures. In addition, hot spot contention in the memory system can be extremely disruptive.

An optical interconnect cannot hope to boost performance substantially beyond what it is with the already

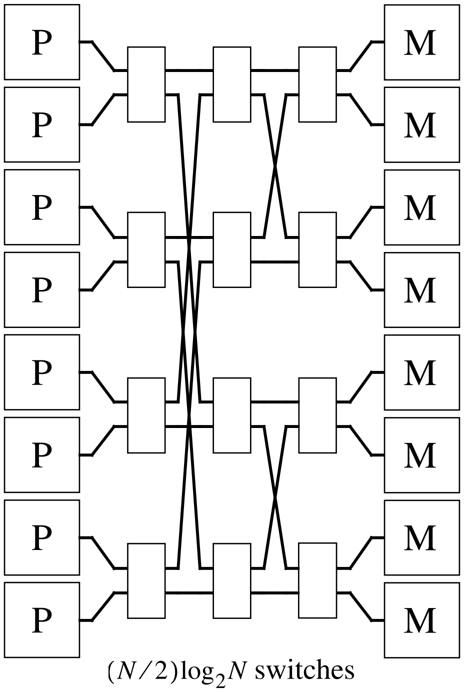

Figure 1. Dancehall architectures

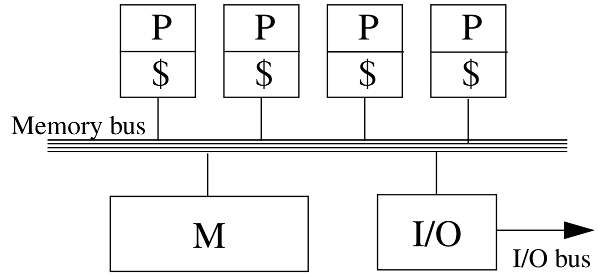

Figure 2. Example 4-node SMP

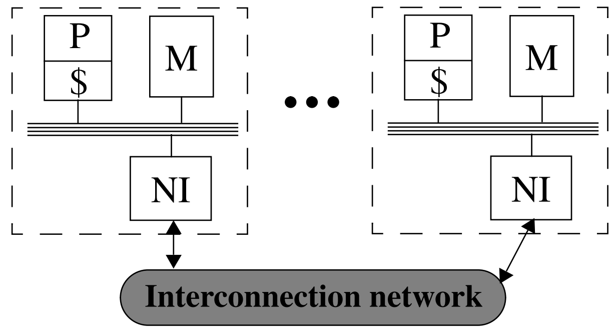

Figure 3. Multicomputer Example

large bandwidths of multistage networks. Furthermore, with processor caches to cut down on memory latency, the optical network would incur the same scalability problems as with the bus-based system described below.

Another class of shared-memory multiprocessor is the bus-based, snooping cache machine (we show an example in Figure 2). This architecture now forms the bulk of all multiprocessor systems sold. In this architecture, the multistage interconnection network is replaced by a single bus. The broadcast capability of the bus is exploited to solve the cache coherence problem by assuring that every processor monitors every memory operation,. Each processor can then intervene when necessary to assure that a single view of memory is maintained.

As processor speeds have increased, fewer processors can be placed on a single bus before the bus becomes saturated. Vendors have managed to place as many as thirty or forty processors on a single bus by pushing the electrical interface to its limit, but this level of parallelism becomes increasingly difficult as ever faster processors emerge.

An optical crossbar or free-space interconnect could function much as an ultra high-bandwidth bus. This is no panacea for scalability of bus-based systems, however. A bus with infinite bandwidth and zero latency would quickly swamp the snooping mechanisms as more processors were added. Because each memory operation must be monitored by every processor, the total processing requirement in the system grows as the square of the number of processors, and neither the bus interface nor the cache tag ports could sustain a linear growth in traffic.

Thus, we see that even the best optical interconnects do not provide a cure-all for shared-memory multiprocessor communication limitations, for either type of shared-memory machine.

2.2 Multicomputers

The incredible performance growth rate of microprocessors, as well as the benefits of economies of scale, have allowed the multicomputer to emerge as the dominant scalable multiprocessor. Workstation-like multicomputer nodes provide low-latency access to a fraction of the system memory (although the two-tiered memory model makes these machines very hard to program). Recent examples of multicomputers include the Thinking Machines CM-5 [14], the Intel Paragon [5], and the IBM SP-2[12]. We show a simple example of a multicomputer architecture in Figure 3.

Multicomputers can provide either a message-passing interface, such as the SP-2, or a shared-memory interface, implemented either with hardware or software sending inter-processor messages and providing the illusion of shared memory (these systems are often called distributed-shared-memory machines).

Not only are the microprocessors off-the-shelf, commodity parts, but increasingly the memory system, interconnect, and communication interface are also exploiting economies of scale. This fact is driving the use of software protocols, such as in the Sequent STiNG [16], and the use of standards, such as the Scalable Coherence Interface [11] in the Convex Exemplar [4].

The emphasis on cost makes it difficult for even an ideal optical network to provide huge performance gains. In message-passing computers, the latency through the actual network wires is typically a small fraction of the total network traversal time; most of the time is spent in the network interface at the sender and receiver. In machines that implement software protocols, such as the Wisconsin Typhoon [18] and Stanford FLASH [13], most of the messaging time is spent running the protocol handlers. Even in hardware-based shared-memory interface systems, such as SCI-based systems or the Stanford DASH [15] machine, the shared-memory protocol logic may be a bottleneck. While support could be added to increase the bandwidth of the protocol engines, the increasing reliance on commodity parts makes adding such support problematic. Again, an ideal optical network achieving infinite bandwidth and zero latency would not improve the performance dramatically for these machines.

If the network interface problems were alleviated, a very low-latency optical network could possibly provide near-uniform access times to all of the memories in the system, whether local or remote. This situation would be transitory at best, however.

As microprocessors become ever faster, long access latencies for even local memories, as well as limited bandwidth off the processor die and on the system bus, will force the local memory progressively closer to the CPU. We believe that this trend will culminate with a node's local memory on the same die (or module) as the processor, exploiting the tremendously high bandwidth and low latency out of on-chip memory banks.

This design point will serve to make remote communication orders of magnitude more expensive than local (on-chip) communication, as the gap between a local on-chip memory access and a remote memory access grows. This gap will in turn make good speedups in massively parallel systems even harder to obtain for codes that are not "embarrassingly parallel."

It is therefore paramount that both sequential portions of the code, including the affiliated communication, be as efficient as possible. It will also be critical that communication during parallel phases be minimized or tolerated by overlapping with computation; otherwise scalability will suffer.

3 Implementing SPSD execution

Our proposal for reducing serial overhead in parallel programs exploits the fact that computation will be significantly cheaper in future systems, particularly those that have tightly-coupled processors and main memories (rendering remote communication more expensive). For the purposes of this discussion, we will assume that every parallel program can be decomposed (albeit at a fine grain) into two modes: serial and parallel.

We can improve both the serial portion of a parallel program, and uniprocessor programs, with an execution model that is analogous to the Single-Program, Multiple Data stream (SPMD) execution model identified by Darema-Rogers *et al.* in 1985 [6]. This execution model, which we call Single-Program, Single Data stream (SPSD), was derived from the Massive Memory Machine work from the early 1980s [8]. In SPSD mode, each of the processors executes the entire program, reading and writing exactly the same data (unlike SPMD, in which each processor writes to different addresses).

SPSD execution was conceived to run on DATASCALAR systems [3], which are small-scale systems that contain processors tightly coupled with main memory, running uniprocessor programs. DATASCALAR architectures are optimized for efficient serial execution with exploitation of coarser-grain parallelism when possible. For an MPP with an optical network, the goal of SPSD execution is precisely the converse: the program runs in parallel mode the majority of the time, switching to SPSD to race through a serial section of code.

Each node assumes ownership of the portion of physical address space that it contains. When a node is operating in SPSD mode, and issues a load to an operand that it owns, it broadcasts that operand to the other nodes (since they are all running the same code, they too will eventually need that operand). When a node issues a load to an operand that a different node owns, the load stalls, if necessary, until the needed operand arrives over the optical network, broadcast by the owning node. This ownership/broadcast scheme was called *ESP* by the Massive Memory Machine work.

To cut down on inter-chip communication, and the latencies associated therewith, we replicate some of the heavily-accessed pages across all nodes. Accesses to this *statically replicated* data will complete locally on every node, not requiring a broadcast. Memory on each node is thus divided into two classes: *replicated* and *communicated*. A load to a *replicated* datum never requires a broadcast since it completes on every node, and a load to a *communicated* datum always requires a broadcast, since it completes only on the node that owns that particular datum. Data may also be replicated dynamically; we allow each node to cache data owned by other nodes. A load to a communicated datum that is found in all processor caches is not broadcast.

In Figure 4 we show how loads and stores to replicated versus communicated memory differ; both CPUs issue a load and store to replicated memory (L1 and S1), which complete on both nodes. Both CPUs also issue commands

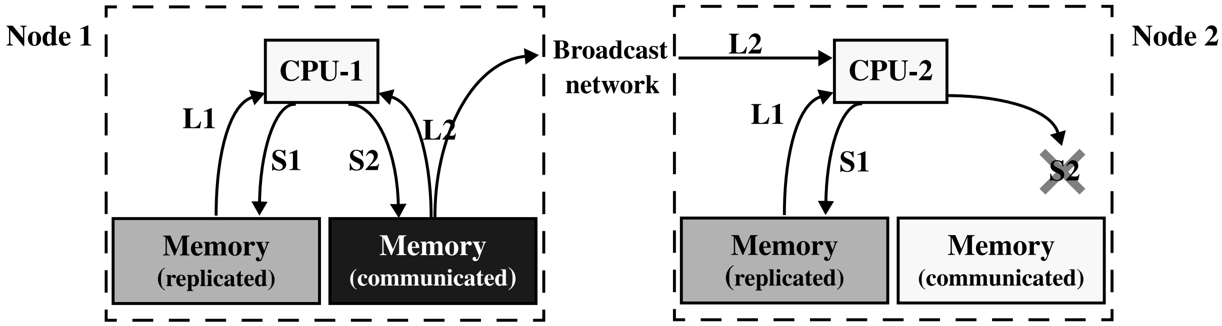

Figure 4. Replicated versus communicated main memory

L2 and S2, which are located in the communicated memory of node 1 only. Node 1 broadcasts L2, which node 2 receives and consumes. S2 completes at node 1, but is dropped at node 2.

The rest of this section describes the three categories of benefits that the SPSD mode of execution provides.

3.1 Request elimination through ESP

The Massive Memory Machine (MMM) defined *ESP*, the notion of running the same program across multiple computational engines, broadcasting accessed local data to all non-local processors. However, the MMM proposed conventional, non-pipelined uniprocessors connected by a single global bus, and was therefore unlikely to provide better cost-performance than competing solutions. Furthermore, the MMM was a fully synchronous architecture, in which all processors proceeded in lock-step, with one processor running slightly ahead of the others (the *lead* processor). In Figure 5a we illustrate the high-level design of the MMM. In Figure 5b we show an example of the MMM's operation, in which processor 3 owns the first four operands, so is the lead processor for the first four accesses. Processor 2 owns operands five through seven, so upon the fifth access, a *lead change* occurs and processor 2 becomes the lead processor. Finally, another lead change occurs on the access to the eighth operand, and processor 3 again becomes the lead processor.

An MPP system running in SPSD mode enjoys the same benefits from ESP as did the MMM proposal. The major benefit in this case is reduced remote access latency, since only one network traversal is needed for a remote operand (as opposed to two for the traditional request/response pair, or more if coherence protocol actions are required). Other related benefits for SPSD are: (1) elimination of interconnect request traffic, and (2) elimination of interconnect write traffic.

Because each node runs the same program, a communicated operand can be sent to the other nodes as soon as its address is resolved and the operand is fetched from the local memory. The request part of the access involves only an on-chip lookup. The operand is sent directly to the other nodes, eliminating half of the communication delay by requiring only one-way communication.

This "response-only" model also reduces traffic (increasing effective off-chip bandwidth) because off-chip requests are unneeded. Finally, all inter-chip write traffic is eliminated under ESP. Stores (or write-backs of dirty cache lines) complete locally on every node if their target address is contained within a replicated page. Stores or write-backs to a communicated page occur only on the owning node, which preserves consistency since that node holds the only copy in main memory. Note that there are no consistency issues, because every node is running the same code.

3.2 Pipelined memory prefetching

Consider an access to a datum obtained through a pointer. In conventional systems: (1) a request must be sent off-chip to memory, (2) the pointer is returned, the processor computes the address of the datum, (3) sends a request to memory, and (4) the operand is returned. This sequence requires a total of four sequential chip-to-chip crossings. An ESP-based system would incur two sequential chip crossings at most: (1) the owner of the pointer broadcasts the address, all nodes compute the address of the datum, and then (2) the owner of the datum broadcasts the datum.

An MPP running in SPSD mode can do even better. If both the pointer and datum reside on the same node—the owner can therefore read both without waiting for an off-chip access, pipelining the broadcast of both operands to the other nodes. We call the phenomenon of multiple consecutive accesses falling on the same node *pipelined memory prefetching*. Since each memory chip has an on-chip processor, consecutive accesses falling on *any* memory chip will cause memory prefetching. Another way of visualizing memory prefetching is from the point of view of one node—from its perspective, it is the processor actually performing the serial phase of the program, and all other nodes are simply memory—which can send it operands that it will need, before it has even computed their addresses.

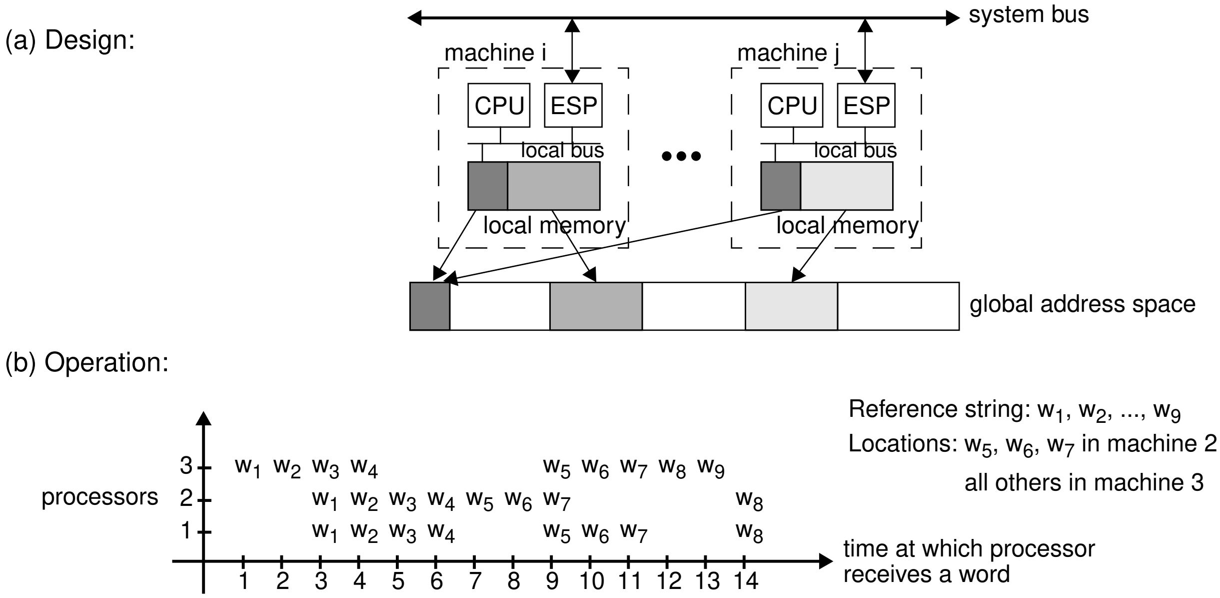

Figure 5. Design and operation of the ESP Massive Memory Machine (taken from [8])

Whenever the address of an operand depends on another operand, and the two operands reside on different nodes, an inter-chip message is required. That communication effectively halts any memory prefetching occurring down that dependence chain on any node. An example can be seen in Figure 5: if each w_{i+1} is dependent on w_i, there are only two inter-chip latencies on the critical path (after accessing w_4 and w_7). To increase the performance gains from memory prefetching, it is therefore desirable to maximize the number of consecutive references on single nodes. We refer to the number of consecutive references to operands on a single node as a *streak*. A streak includes both replicated and communicated references.

With an in-order issue processor, a break in a streak will force the node to stall until another node broadcasts the needed operand. An out-of-order issue machine lends itself particularly well to this model, however, as multiple nodes may simultaneously prefetch down distinct dependence chains if the instruction window is sufficiently large. The ideal case is where all nodes are memory prefetching down separate dependence chains that they contain locally.

Memory prefetching does not require software support or re-compilation—running SPSD in a parallel system may exploit spatial locality already inherent in reference streams. (Programs may benefit from re-compilation or programmer tuning, of course, since explicit support could increase average streak length.) When streaks are greater than average, the SPSD model benefits, since inter-chip latencies on the critical path are reduced.

3.3 New opportunities for parallelism

When communication becomes relatively much more expensive than computation, performing redundant computation to avoid communication becomes more attractive. SPSD uses redundant computation to reduce remote latencies. We can also use SPSD to extract new forms of "memory parallelism" that further reduce serial code.

For example, assume that a chained hash table is distributed across the physical memory of multiple nodes. We modify the run-time storage allocator to place any chained elements on the same node as the head of the chain. Because this is code that is hard to -parallelize, we are running in SPSD mode. All nodes thus compute the index of the array when performing a hash table lookup, insertion, or deletion.

Serial overhead is reduced by placing a "locality branch" around the hash table operation. When performing insertions and deletions, only the node containing the chain performs the operation, with *no remote communication*. For a lookup, the owner broadcasts the result of the lookup, without any of the intervening chain addresses or keys.

Many other examples of such opportunities under SPSD exist, but are beyond the scope of this paper. More detail can be found elsewhere [3].

3.4 SPSD and optical interconnects

The major drawback of the SPSD model is that it requires broadcast of data to all nodes. This limits its appeal using traditional electrical interconnects to a small number of nodes. To be successful, the SPSD model requires inexpensive broadcasts, which makes it a good match for optics. Since all nodes on a time-multiplexed or wave-multiplexed optical interconnect can see any channel they choose (*any*, but not *all*), broadcasts essentially come for free. With electrical interconnects, SPSD is limited to small-scale systems that also have low-overhead broadcasts (e.g., buses and small rings).

Furthermore, the optical interconnect can multicast to subsets of the nodes in an MPP, and in fact can simultaneously multicast different data to different subsets. This capability may eventually prove useful for more creative and flexible ways of reducing communication, both in parallel and serial phases of the program's execution.

The need for flow control is greatly mitigated with the SPSD model. Since the processors are all executing the same code, the number of communications depends not on the number of nodes, but the number of operands being communicated. This number will grow much more slowly, since more nodes means more of the data can be replicated. In addition, hot spots are less likely, because the single program execution effectively produces its own flow control.

3.5 SPSD implementation issues

Because all operands for the serial phases must be present at every processor, running in SPSD mode can only be more efficient if most accesses can be found locally, reducing the average local memory access latency. It is well known, however, that a large majority of memory references tend to access a small minority of the memory locations. For this reason, cache memories—particularly those specifically designed with this in mind—are often able to reduce remote accesses, sometimes dramatically [2, 9]. While static replication of small numbers of "hot pages" can cut down substantially on remote accesses, dynamic replication—achieved with cache memories—can reduce remote accesses even further. Using caches to reduce the number of broadcasts introduces the problem of keeping caches across nodes *correspondent*; the details of the solution are beyond the scope of this paper and appear elsewhere [3].

Speculative execution also complicates this model. Speculative code resulting from branch prediction can either hold onto broadcasts until the branch target is resolved, or speculatively broadcast communicated operands with some sort of sophisticated tagging scheme. Coarse-grain speculative processors must guarantee that large-scale speculative tasks issue the same way across nodes.

One drawback to using SPSD execution in massively parallel processors is the reliance on commodity microprocessors in such systems. It is difficult to justify architectural changes to support a market as small as MPPs. However, SPSD was originally envisioned for aggressive uniprocessor systems[2]. The success of such small-scale systems might well produce much of the required support on future microprocessors (e.g., queues for buffering broadcasts and matching them with processor requests).

4 Looking into the future

Optical interconnects provide a host of interesting opportunities for the development of future massively parallel systems. However, economies of scale—coupled with architectures balanced and tuned for electrical interconnects—will force optics to demonstrate a quantum leap in capability before their widespread adoption.

The SPSD execution model is particularly appealing because the computer appears to be a conventional uniprocessor. We envision that a computer employing the SPSD model will run programs not explicitly written for it, with the compiler discovering and exploiting the obvious parallelism of the program. The DATASCALAR architecture can easily switch between the conventional parallel (MIMD) model and the SPSD model, exploiting the compiler-discovered parallelism as appropriate, yet achieving very high performance on the portions not easily parallelized. In addition, with the help of the programmer (or a sophisticated compiler) further optimizations are possible to capture opportunities for memory parallelism. The DATASCALAR model appears promising for semiconductor-based technologies just over the horizon, when processing power is readily available where needed. With semiconductor interconnects, however, the scalability is seriously limited by the requirement of broadcast, and in fact we do not envision DATASCALAR systems beyond twenty or so separate modules. These systems, of course, might be used as components in larger systems. The opportunity for low-latency, scalable broadcast using free-space optical interconnects promises the potential for much larger systems. How far such systems could be extended before other factors limit their scalability is a subject of future research.

For the SPSD model, adding more processors can be effective for applications where the data set grows without increases in computation. Problems that require more memory, but do not require more computation, can be accommodated by larger numbers of modules. This model scales from the communications standpoint because the program execution inherently provides flow control that limits the emergence of hot spots.

How far the SPSD model can scale depends on the extent to which communications can be limited. The use of replicated data may result in one to two orders of magnitude reduction in traffic. Other optimizations are also possible.

5 Summary

In this paper we have presented an execution model, adapted from an aggressive uniprocessor proposal, which exploits the cheap broadcasting capability of optical networks to reduce sequential overheads. Optical interconnects will not become the communications method of choice unless they are able to demonstrate clearly superior capabilities. To do this, the architecture must exploit the unique opportunities that the technology offers, not simply settle for a higher-bandwidth communications network. Novel architectures are needed to take advantage of the benefits offered by optical technology. We have demonstrated one such architecture and given arguments for why the DATASCALAR architecture is well matched for optical networks in large-scale, high-performance systems of the future.

Acknowledgments

The authors thank Allan Gottlieb for the opportunity to present our viewpoints in this forum. We thank Leon McCaughan for his discussions on optical technologies, and the other members of the Galileo project, Stefanos Kaxiras and Alain Kägi, for their contributions to the ideas presented in this paper.

References

[1] John Beetem, Monty Denneau, and Don Weingarten. The GF11 Supercomputer. In *ISCA12*, pages 108–115.

[2] Doug Burger, James R. Goodman, and Alain Kägi. Memory Bandwidth Limitations of Future Microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 79–90, May 1996.

[3] Doug Burger, Stefanos Kaxiras, and James R. Goodman. DataScalar Architectures and the SPSD Execution Model. Technical Report 1317, Computer Sciences Department, University of Wisconsin, Madison, WI, June 1996.

[4] Convex Computer Corporation, Richardson, Texas. *SPP1000 Systems Overview*, 1994.

[5] Intel Corporation. Paragon Technical Summary. Intel Supercomputer Systems Division, 1993.

[6] F. Darema-Rogers, V. A. Norton, and G. F. Pfister. Using a Single-Program Multiple-Data Computation Model for Parallel Execution of Scientific Applications. IBM Research Report RC 11552, November 1985.

[7] Michael J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, C-21:948–960, 1972.

[8] Hector Garcia-Molina, Richard J. Lipton, and Jacobo Valdes. A Massive Memory Machine. *IEEE Transactions on Computers*, C-33(5):391–399, May 1984.

[9] James R. Goodman. Using Cache Memory To Reduce Processor-Memory Traffic. In *Proceedings of the 10th Annual International Symposium on Computer Architecture*, pages 124–131, June 1983.

[10] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir. The NYU Ultracomputer–Designing an MIMD Shared Memory Parallel Computer. *IEEE Transactions on Computers*, C-32(2):175–189, February 1983.

[11] IEEE Computer Society. Scalable Coherent Interface (SCI). ANSI/IEEE Std 1596-1992, August 1993.

[12] Inc. International Business Machines. IBM RISC System/6000 Scalable POWERparallel Systems: Command and Technical Reference. Reference Manual, 1993. GC23-3900-00.

[13] Jeffrey Kuskin, David Ofelt, Mark Heinrich, John Heinlein, Richard Simoni, Kourosh Gharachorloo, John Chapin, David Nakahira, Joel Baxter, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John Hennessy. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 302–313, April 1994.

[14] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St. Pierre, David S. Wells, Monica C. Wong, Shaw-Wen Yang, and Robert Zak. The Network Architecture of the Connection Machine CM-5. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, June 1992.

[15] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica Lam. The Stanford DASH Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[16] Tom Lovett and Russell Clapp. STiNG: A CC-NUMA Computer System for the Commercial Marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 304–315, May 1996.

[17] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss. The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 764–771, August 1985.

[18] Steven K. Reinhardt, James L. Larus, and David A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 24–33, April 1994.