

Analysis and Modeling of Advanced PIM Architecture Design Tradeoffs*

Ed Upchurch
Thomas Sterling
California Institute of Technology

Jay Brockman
University of Notre Dame

October 6, 2004

1 Introduction

A major trend in high performance computer architecture over the last two decades is the migration of memory in the form of high speed caches onto the microprocessor semiconductor die. Where temporal locality in the computation is high, caches prove very effective at hiding memory access latency and contention for communication resources. However where temporal locality is absent, caches may exhibit low hit rates resulting in poor operational efficiency. Vector computing exploiting pipelined arithmetic units and memory access address this challenge for certain forms of data access patterns, for example involving long contiguous data sets exhibiting high spatial locality. But for many advanced applications for science, technology, and national security at least some data access patterns are not consistent to the restricted forms well handled by either caches or vector processing. An important alternative is the reverse strategy; that of migrating logic in to the main memory (DRAM) and performing those operations directly on the data stored there. Processor in Memory (PIM) architecture has advanced to the point where it may fill this role and provide an important new mechanism for improving performance and efficiency of future supercomputers for a broad range of applications. One important project considering both the role of PIM in supercomputer architecture and the design of such PIM components is the Cray Cascade Project sponsored by the DARPA High Productivity Computing Program. Cascade is a Petaflops scale computer targeted for deployment at the end of the decade that merges the raw speed of an advanced custom vector architecture with the high memory bandwidth processing delivered by an innovative class of PIM architecture. The work represented here was performed under the Cascade project to explore critical design space issues that will determine the value of PIM in supercomputers and contribute to the optimization of its design. But this work also has strong relevance to hybrid systems comprising a combination of conventional microprocessors and advanced PIM based intelligent main memory.

Processor in Memory or PIM architecture incorporates arithmetic units and control logic directly on the semiconductor memory die to provide direct access to the data in the wide row buffer

*0-7695-2153-3/04 \$20.00 (c)2004 IEEE

of the memory. PIM offers the promise of superior performance for certain classes of data intensive computing through a significant reduction in access latency, a dramatic increase in available memory bandwidth, and expansion of the hardware parallelism for concurrency of flow control. Advances in PIM architecture under development incorporate innovative concepts to deliver high performance and efficiency in the presence of low data locality. These include the use of PIM to augment and compliment conventional microprocessor architectures, the use of a large number of on-chip PIM nodes to expose a high degree of memory bandwidth, and the use of message-driven computation with a transaction-oriented producer-consumer execution model for system-wide latency tolerance. All of these have benefited from previous work and this study extends those experiences to the domain of PIM. The results of these studies have a direct impact on the design of both the advanced PIM architectures being contemplated and the systems in which they are to be incorporated like Cascade. This paper explores the design space of two innovations being considered for PIM through a set of statistical steady-state parametric models that are investigated by means of queuing simulation and analyses.

While the advanced PIM concept is encouraging, it is not proven. In order to both prove the effectiveness of this new class of architecture and to quantitatively characterize the design tradeoff space to enable informed choices of resource allocation, a set of simulation experiments and analytical studies were conducted. These include 1) the modeling of the interrelationship between the PIM components and their host microprocessor, and 2) an exploration of the global latency hiding properties of parcels between PIM devices.

2 Background

Processing-in-memory encompasses a range of techniques for driving computation in a memory system. This involves not only the design of processor architectures and microarchitectures appropriate to the properties of on-chip memory, but also execution models and communication protocols for initiating and sustaining memory-based program execution.

2.1 Reclaiming the Hidden Bandwidth

The key architectural feature of on-chip memory is the extremely high bandwidth that it provides. A single DRAM macro is typically organized in rows with 2048 bits each. During a read operation, an entire row is latched in a digital row buffer just after the analog sense amplifiers. Once latched, data can be paged out of the row buffer to the processing logic in *wide words* of typically 256 bits. Assuming a very conservative row access time of 20 ns and a page access time of 2 ns, a single on-chip DRAM macro could sustain a bandwidth of over 50 Gbit/s. The memory capacity on a single PIM chip may be partitioned into many separate memory banks, each with its own arithmetic and control logic. Each such bank, or node, is capable of independent and concurrent action thereby enabling an on-chip peak memory bandwidth proportional to the number of such nodes. Using current technology, an on-chip peak memory bandwidth of greater than 1 Tbit/s is possible per chip. A typical memory system comprises multiple DRAM components and the peak memory bandwidth made available through PIM is proportional to this number of chips. Much of PIM research has focused upon reclaiming this hidden bandwidth, either through new organization for conventional architectures or through custom ISAs.

Several studies have demonstrated that simple caches designed for on-chip DRAM can yield performance comparable to classical memory hierarchies, but with much less silicon area. In [28], researchers at Sun investigated the performance of very wide, but shallow caches that transfer an entire cache line in a single cycle. Using a Petri-net model, they showed that as a result of

the lower miss penalty, a PIM with a simple 5-stage RISC pipeline running at 200 MHz would have comparable performance to a DEC Alpha 21164 running at 300 MHz, with less than one-tenth the silicon area. Work at Notre Dame showed similar performance results for a sector cache implemented by adding tag bits directly to the row buffers in DRAM [3]. Early simulation results from the Berkeley IRAM project showed that in addition to improved performance-per-area, PIM could also have much lower energy consumption than conventional organizations [12].

Even greater performance gains are possible through architectures that perform operations on multiple data words accessed from memory simultaneously. Many such designs have been implemented or proposed [5][12] [13][21] [19][22]. The Berkeley VIRAM has 13 Mbytes of DRAM, a 64-bit MIPS scalar core, and a vector coprocessor with 2 pipelined arithmetic units with each organized into 4 parallel vector lanes. VIRAM has a peak floating-point performance of 1.6 Gflop/s, and shows significant performance improvements in multimedia applications over contemporary superscalar, VLIW, and DSP processors [21]. The DIVA PIM employs a wide-word coprocessor unit supporting SIMD operations similar to the Intel MMX or PowerPC AltiVec extensions. Using a memory system enhanced with DIVA PIMs produced average speedups of 3.3 over host-only execution for a suite of data-intensive benchmark programs [13]. Memory manufacturer Micron's Yukon chip is a 16 Mbyte DRAM with a SIMD array of 256 8-bit integer ALUs that can sustain an internal memory bandwidth of 25.6 Gbytes/s [19].

2.2 Computation and Communication in Massively-Parallel PIM Systems

The benefits of PIM technology can be further exploited by building massively-parallel systems with large numbers of independent PIM nodes (an integrated memory/processor/net-working device). Many examples of fine-grain MPPs have been proposed, designed and implemented in the past—for example [15] and others. All have faced stiff challenges in sustaining significant percentages of peak performance related to the interaction of computation and communication, and there is no reason to assume that networked PIM devices would be immune to the same problems. PIM does, however, provide a technology for building massive, scalable systems at lower cost, and for implementing highly efficient mechanisms for coordinating computation and communication.

One of the key potential cost advantages of PIM is the ability to reduce the overhead related to memory hierarchies. In [23], it was first suggested that a petaflops scale computer could be implemented with a far lower chip count using PIM technology than through a network of traditional shared memory machines or through a cluster of conventional workstations. The J-Machine was one of the computers envisioned as using DRAM based PIM components for an MPP, although for engineering considerations the system was eventually implemented in SRAM technology [10]. Execube [20] was the first true MIMD PIM, with 8 independent processors connected in a binary hypercube on a single chip together with DRAM. More recently, IBM's original Blue Gene [11] and current BG/L designs [1] both use embedded DRAM technology in components for highly scalable systems.

Although related, the semantics of requests made of a PIM system differ somewhat from messages in classic parallel architectures. HTMT and related projects introduced the concept of *parcels* (parallel control elements) for memory-borne messages, which range from simple memory reads and writes, through atomic arithmetic memory operations, to remote method invocations on objects in memory [31][5].

There are various ways that one could characterize and set performance objectives for PIM networks communicating through parcels. A useful approach is to view the PIM network as a fine grain transaction-processing system, where two important, related figures of merit are the latency in servicing a single transaction and the throughput, or number of transactions serviced

per unit time. As with fine-grain MPPs, the keys to performance for PIM systems are minimizing the overhead of context switching and communication, and overlapping communication and useful computation wherever possible. Several projects have addressed hardware support for low-overhead communication, notably the MDP [9] and J-Machine [25]. Active messages [32] is a software solution that minimizes communication overhead by including the address of a user-level routing with the message that efficiently unpacks the message and integrates it into ongoing computation. A variety of architectures and execution models have also addressed support for overlapping computation and communication, that all entail mechanisms for scheduling operations out of a pool of available work. These include dataflow machines [2] [26], multithreading [30] [29] [7], and hybrids [16] [24]. PIM Lite is a recent PIM architecture and prototype implementation that efficiently uses wide words out of memory to integrate multithreading and fast parcel response with SIMD arithmetic operations [5] [4].

2.3 A Quantitative Framework for Design Space Exploration

As the previous sections show, many architectural and implementation options currently exist for exploiting PIM technology. What is lacking, however, is a systematic quantitative evaluation of the tradeoffs among the design alternatives to achieve balanced, cost-effective systems: what follows is a set of key statistical parametric studies to provide a quantitative framework for assessing the tradeoff space defined by critical design factors. Specifically, we have developed a set of analytic and simulation models that help provide insight into some of the key questions affecting the configuration of PIM systems.

- The first set of analyses addresses tradeoffs in partitioning a computation into heavyweight/high temporal locality threads running on a conventional host processor and light-weight/low temporal locality threads running in PIM. Parameters of the model include the number of PIM nodes, the percentage of the application with low temporal locality, and the system configuration.
- The second set of analyses investigates how effectively parcels can hide latency (or overlap communication and computation). This work is related to prior work in studying the effectiveness of multithreading [27], but is set in a PIM context.

3 PIM-based System Simulation Results

Temporal locality is an important property of computations that reflects the data reuse by processing elements when accessed from main memory. Cache hit rates and register pressure are very sensitive to temporal locality. When it is high, processors based on caches and optimized for high clock rates perform well. When it is low, the same computing elements may exhibit performance efficiencies measured in single digits. Conversely, PIM may perform relatively well for low temporal locality computation because they have short latency of access between local logic and memory row buffers, and because they may provide very high memory bandwidth, which with sufficient parallelism may deliver high memory throughput even with low temporal locality. Therefore, a set of experiments were devised to expose the tradeoff space between high speed cache based processors that we will refer to here as heavyweight processors (HWP) and high bandwidth low latency processors we will refer to here as lightweight processors (LWP). The primary independent variable of these tradeoff studies is a measure of the PIM workload which reflects temporal locality. When

Parameter	Description	Experimental Value
W	total work = $W_H + W_L$	100,000,000 operations
$\%W_H$	percent heavyweight work	varied 0% to 100%
$\%W_L$	percent lightweight work	varied 0% to 100%
T_{Hcycle}	heavyweight cycle time	1 nsec
T_{Lcycle}	lightweight cycle time	5 nsec
T_{MH}	heavyweight memory access time	90 cycles
T_{CH}	heavyweight cache access time	2 cycles
T_{ML}	lightweight memory access time	30 cycles
P_{miss}	heavyweight cache miss rate	0.1
$mix_{l/s}$	instruction mix for load and store ops	0.30

Table 1: Parametric Assumptions and Metrics

on a cache miss. The LWP has no cache but is physically adjacent to the memory row buffer and so exhibits much shorter memory access times.

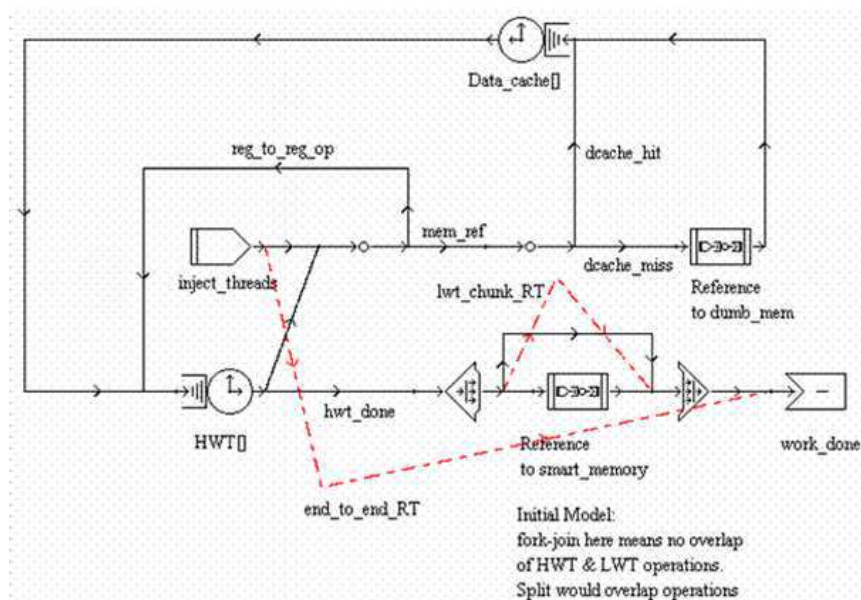


Figure 2: SES Queuing Model of Heavyweight Processor

Figure 2 presents the simple queue model for the HWP and Figure 3 provides the corresponding queue model for the array of LWP and memories. Note that for simplicity, the model treats the main memory accessed by the HWP and LWP as separate devices but this is simply an artifact of convenience and does not impact the simulation results. Bank conflicts are not modeled but the nature of the workload modeled for these experiments precludes this kind of resource contention so no inaccuracies are introduced in the final results.

The experimental workload divides the operations between the HWP and the array of LWP. For those threads of activity that exhibit high temporal locality such that good cache hit rates should be expected, the HWP is scheduled to perform them. For those threads of activity that exhibit low or no temporal locality that would result in very poor cache performance, the set of LWP/memory components are scheduled to perform them. At any one time, either the HWP or LWP array is executing but not both. We also assume that the LWP workload is partitionable in to a number of

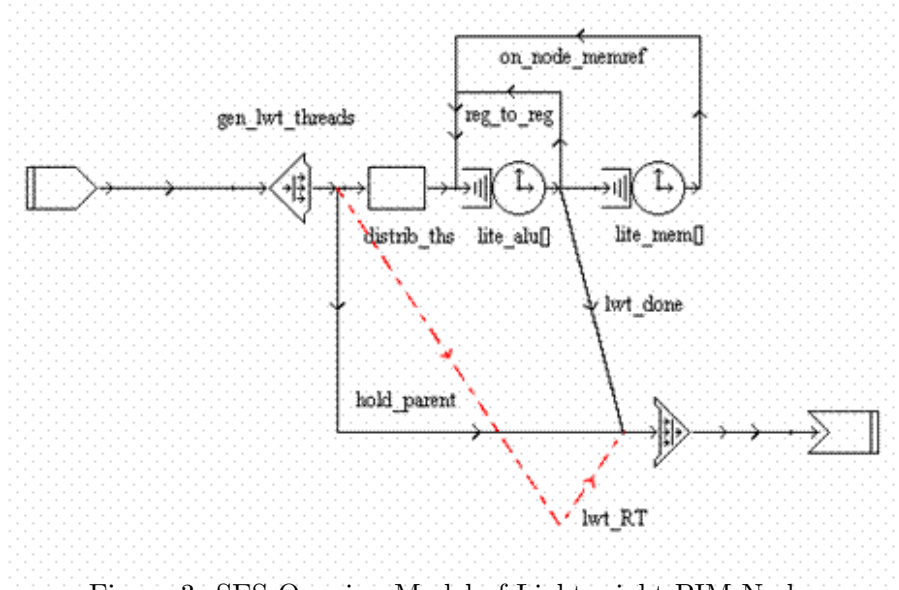


Figure 3: SES Queuing Model of Lightweight PIM Nodes

concurrent threads that are concurrent and uniform in length, one per LWP. This execution flow is depicted in Figure 4.

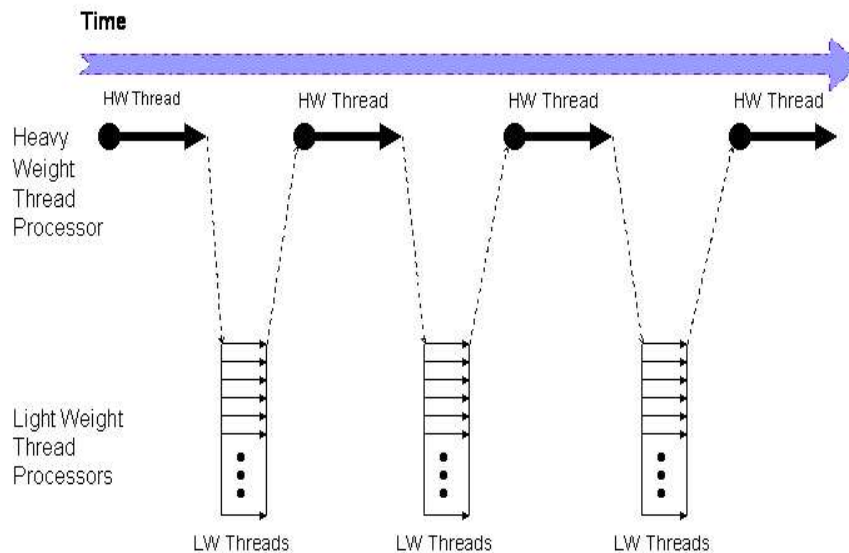


Figure 4: Threads Timeline

While somewhat constraining, the experimental workload permits simple statistical characterization and is representative of many important classes of real-world algorithmic behavior if by no means all. Most regular problems fall in to this category and are programmed this way with such conventional programming models as MPI. The parameters used to specify the workload are also given in Table 1.

3.1.1 Experimental Results from the Queuing Simulation

Two experiments were performed: 1) a control run in which the HWP performed all of the work, and 2) the test runs in which the low locality threads were performed on a set of LWP nodes. For both cases, the amount of low locality work measured as the percentage of operations was varied across a parameter range of between 0% and 100%. For the test runs, the number of LWP nodes was varied as well in a range typical of a modest scale system. The performance gains of the test runs with respect to the control runs were calculated as a function of the fraction of LWP workload for different number of LWP nodes as shown in Figure 5.

It is seen that even for a small amount of LWP work including PIMs in the system may double the performance. If the application is data intensive, a significant portion of the total work is scheduled on the array of LWP nodes and as much as an order of magnitude performance gain may be achieved. In the extreme case where essentially all work resides on the LWP array, at least for some configurations, a factor of 100X gain is observed. These results, if substantiated through further studies, imply important advantages of PIM-based systems with respect to their conventional counterparts. These results are consistent with point studies for specific applications conducted by previously reference projects such as VIRAM and DIVA.

3.1.2 Analytical Model of PIM-based Operation

To better understand the simulated results, an analytical model was developed incorporating the same operational parameters. The results derived from the simulation, shown in Figure 6, were reproduced with this analytical model to an accuracy of between 5% and 18%. This encouraging result motivated a second analytical study to expose the basic time to solution normalized to that of the HWP alone performing only high temporal locality work; i.e. 0% LWP workload. The equations are given below:

$$Time_{relative} = 1 - \%W_L \times \left\{ 1 - \frac{1}{N} \times \left[\frac{T_{Lcycle} + mix_{l/s} \times (T_{ML} - T_{Lcycle})}{1 + mix_{l/s} \times (T_{CH} - 1 + P_{miss} \times T_{MH})} \right] \right\}$$

$$\text{let } N_B \equiv \left[\frac{T_{Lcycle} + mix_{l/s} \times (T_{ML} - T_{Lcycle})}{1 + mix_{l/s} \times (T_{CH} - 1 + P_{miss} \times T_{MH})} \right]$$

$$\text{then } Time_{relative} = 1 - \%W_L \times \left\{ 1 - \frac{N_B}{N} \right\}$$

and parameters $\%W_L$, N , and N_B are independent.

This formulation exposes a remarkable property. Totally unanticipated, in addition to the two independent parameters of number of nodes (N) and percentage of LWP workload ($\%W_L$), a third orthogonal parameter, here referred to as N_B , was derived from the combined properties of the system configuration and application workload. This theoretical model is plotted in Figure 7.

From this diagram, it is evident that a point of coincidence occurs at a specific value of N , independent of $\%W_L$. The derived equation for N_B also shows that it is orthogonal to N . For $N > N_B$, time to solution with PIM support will always be as good or better than the control system without PIM elements. If the form of this relationship is sustained as the underlying model grows in fidelity, the finding will provide a strong condition for superiority of PIM-based system architecture.

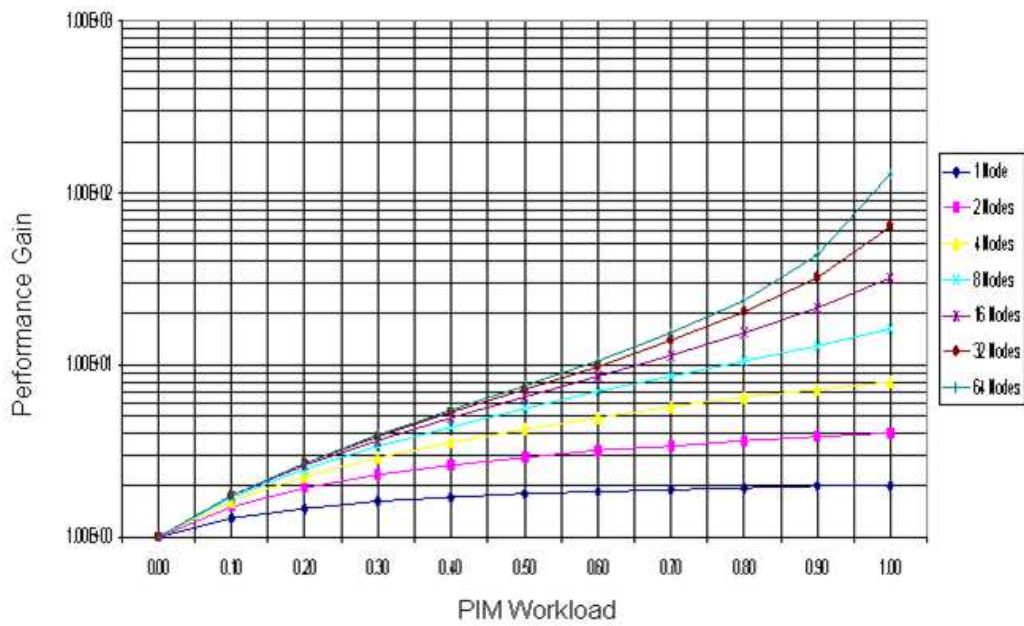


Figure 5: Simulation of Performance Gain

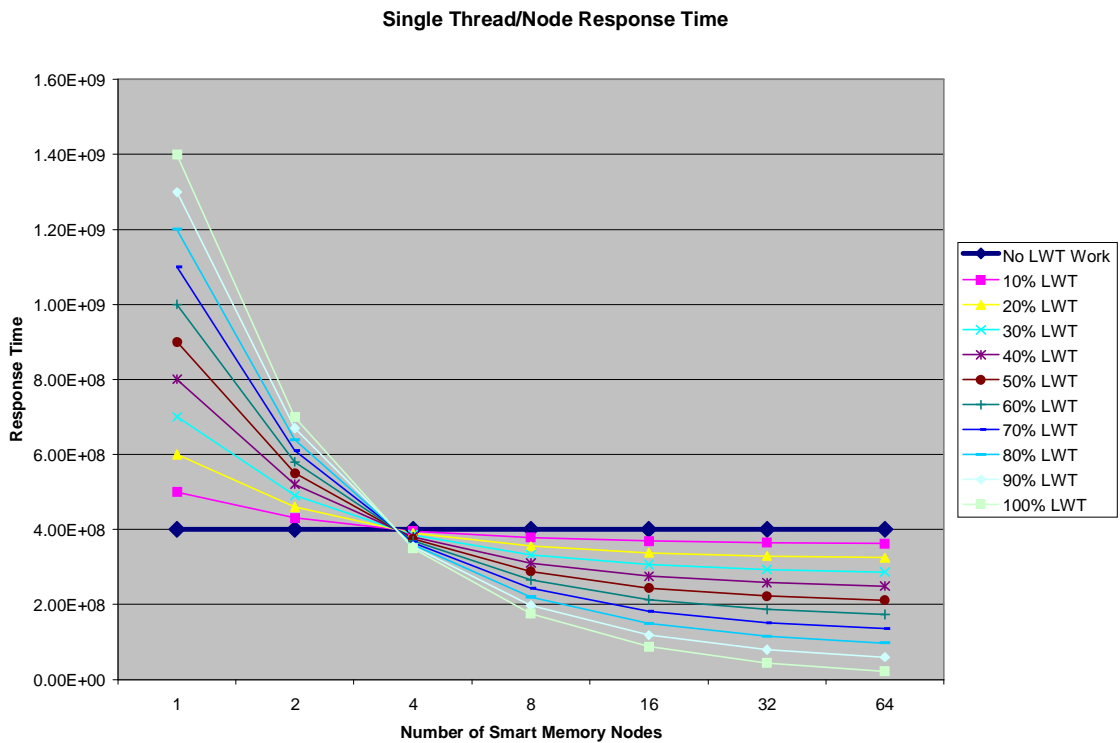


Figure 6: Effect of PIM on Execution Time with Unnormalized Runtime – Simulation Results

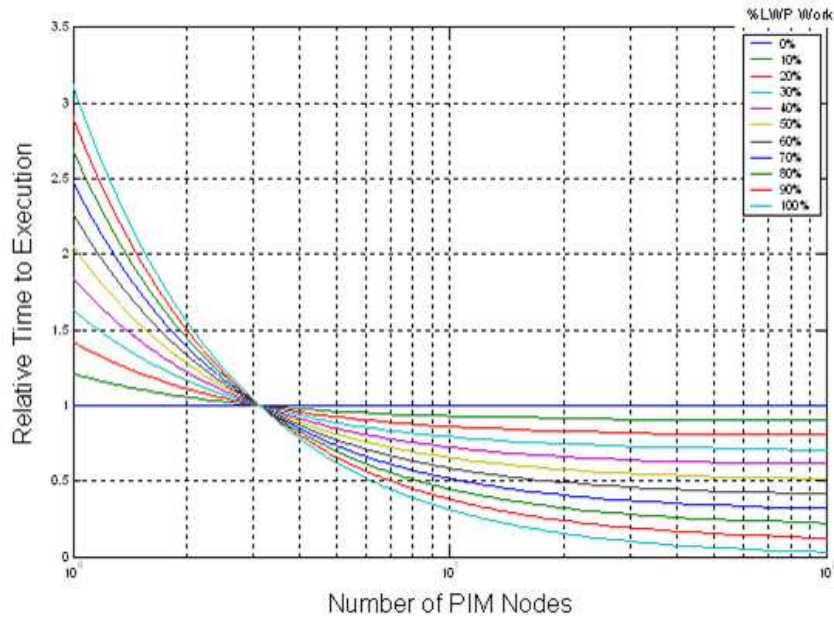


Figure 7: Effect of PIM on Execution Time with Normalized Runtime

4 Parcel-driven Computing Simulation

4.1 Latency Hiding through Parcel Split-transaction Processing

An array of advanced PIM components should logically comprise a single global memory, albeit with non-uniform memory access times, such that any PIM node of one chip can directly reference any physical (or virtual) datum in the memory of any other PIM chip in the array. For very large systems of logically related PIM chips, latency for remote accesses across such an array of PIMs could impose a severe source of performance degradation as it does in certain cases for conventional commodity clusters and MPP systems. This is a consequence of the waiting time experienced by an execution site for the requested data or service.



Figure 8: Parcel Structure

Among the alternatives for latency hiding, hardware supported message-driven computation may provide a particularly attractive approach. In general message driven computing is a protocol by which a remote action is invoked in response to the incidence of a class of message that includes a specification for the action to be performed at the remote site. Even conventional distributed shared memory systems incorporate rudimentary message-driven computing mechanisms in support of compound atomic operations. The concept of message driven computing extends back at least to the 1970's with Hewitt's Actors Model [14] and the various early Data Flow computing models. More recently, Dally's J-Machine at MIT [8] and Culler's Active Messages [32] (software implemented) represent some examples among others of message-driven computing.

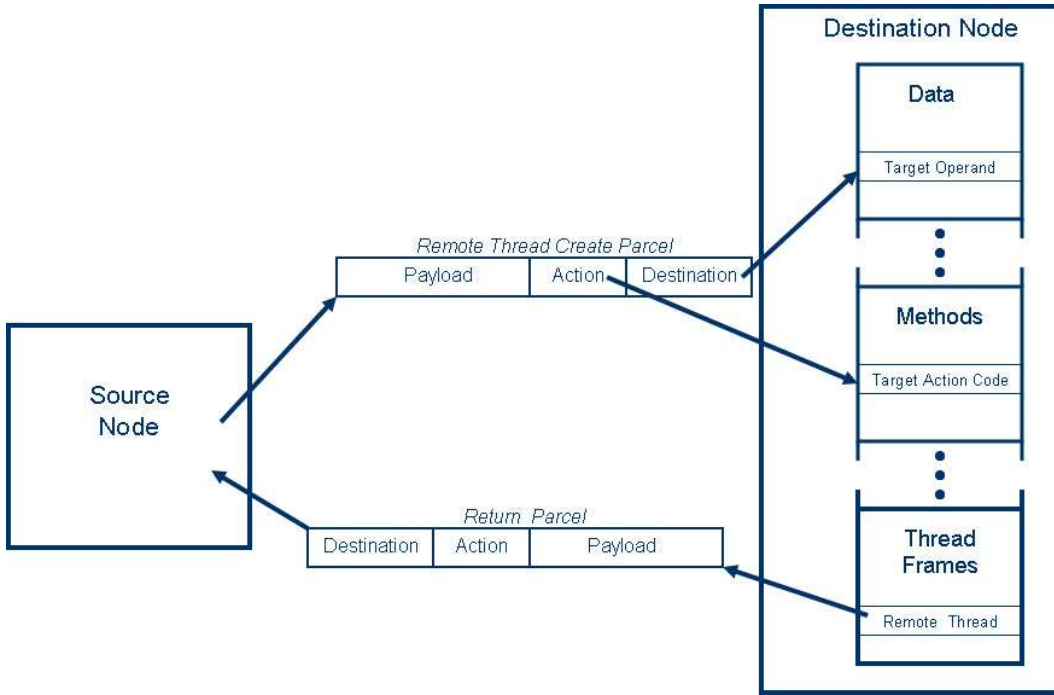


Figure 9: Parcels invoke remote threads

A class of hardware assisted message-driven computing referred to as *parcels* has been investigated by several institutions and projects. Parcels (PARallel Control ELEMENTs) in this context are messages that specify actions to be performed on data elements or data objects in virtual memory. The actions may be simple hardware supported functions or complex functions specified by code blocks. A typical parcel structure is shown in Figure 8 with the outer wrapper employed by the interconnection network transport layer and the inner message providing information including destination data virtual address, action specifier, and additional operand values. Hardware support for parcels minimize overhead of parcel creation, transport, and assimilation including action instantiation. Parcels enable lightweight transaction processing in which an execution site processes incident parcel requests, performs the specified actions locally potentially committing local side-effects, and generating new outgoing parcels to remote data in response. This form of processing is illustrated in Figure 9 where the source node sends a parcel to a data element at a remote node. The parcel both identifies the data object at the remote node upon which to operate and specifies the action to be performed which can be a pointer to a method code block. After performing this action, the remote node in this example returns a result value to the originating source node, although this is not always necessary. In a transaction mode of operation, the execution site does not wait for a response from a remote action as long as there are other parcels waiting to be processed locally. This is sometimes referred to as split transaction execution.

4.2 Parcel Parametric Trade-off Space Experiments

Parcels and split transaction processing may provide a powerful means of hiding system-wide latency. To understand the trade-off space issues of employing parcels and to quantify the potential benefits, a series of statistical parametric studies were performed. These studies, while motivated by the authors' interest in the application of parcels to advanced PIM architecture, are not lim-

ited to PIM but generally apply to systems capable of supporting lightweight split-transaction message-driven computation.

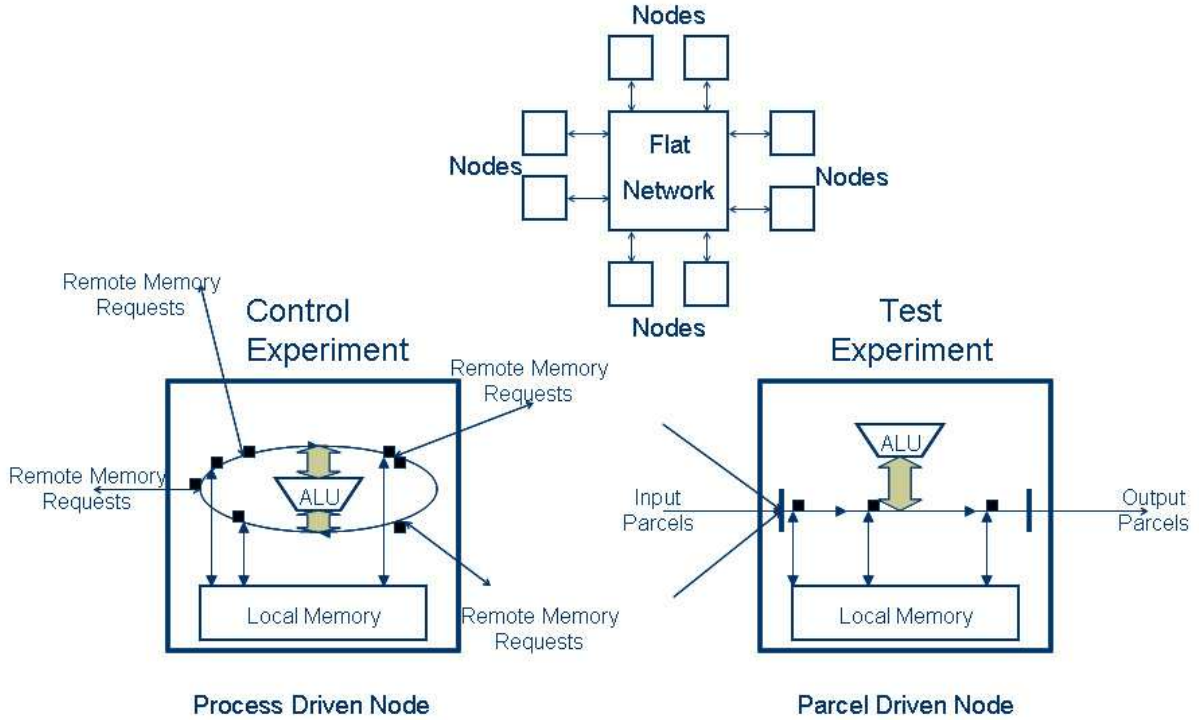


Figure 10: Message Passing versus Parcel-driven lightweight transaction processing

Two systems, a test system and a control system, were simulated using queuing models as shown in Figure 10. The control system provides a basis of comparison for the test system and represents a set of conventional message passing processors. Each processor is in one of three states: performing useful operations, performing local memory access, or waiting for a response to a message it has sent. In this third state, the processor is considered to be idle. The test system operates in split-transaction mode, accepting and processing parcels. Each processor in this model also operates in three states: performing useful operations servicing an active parcel, performing local memory access also on behalf of an active parcel, or idle due to an absence of active parcels to service. All related parameter values are assumed to be the same between these two models including clock rate, peak instruction issue rate, instruction mix, system wide latency which is considered to be flat (fixed delay) for this study, and the degree of remote accesses.

The independent parameters of interest are the degree of parallelism exposed by the split-transaction model, the latency time for remote requests, and the percentage of memory accesses that are remote. The dependent parameters of interest to this study are the relative performance gain of the test system to the control system, and the idle times of both systems. The experiments of both systems are run for the same amount of simulated time and the number of useful operations and local memory access operations, representing the total work done, are measured and compared.

4.3 Parcel Experimental Results

The results of the parcel experiments are presented in Figures 11 and 12. The first shows the results from six major experiments differing in terms of the amount of parallelism available to test

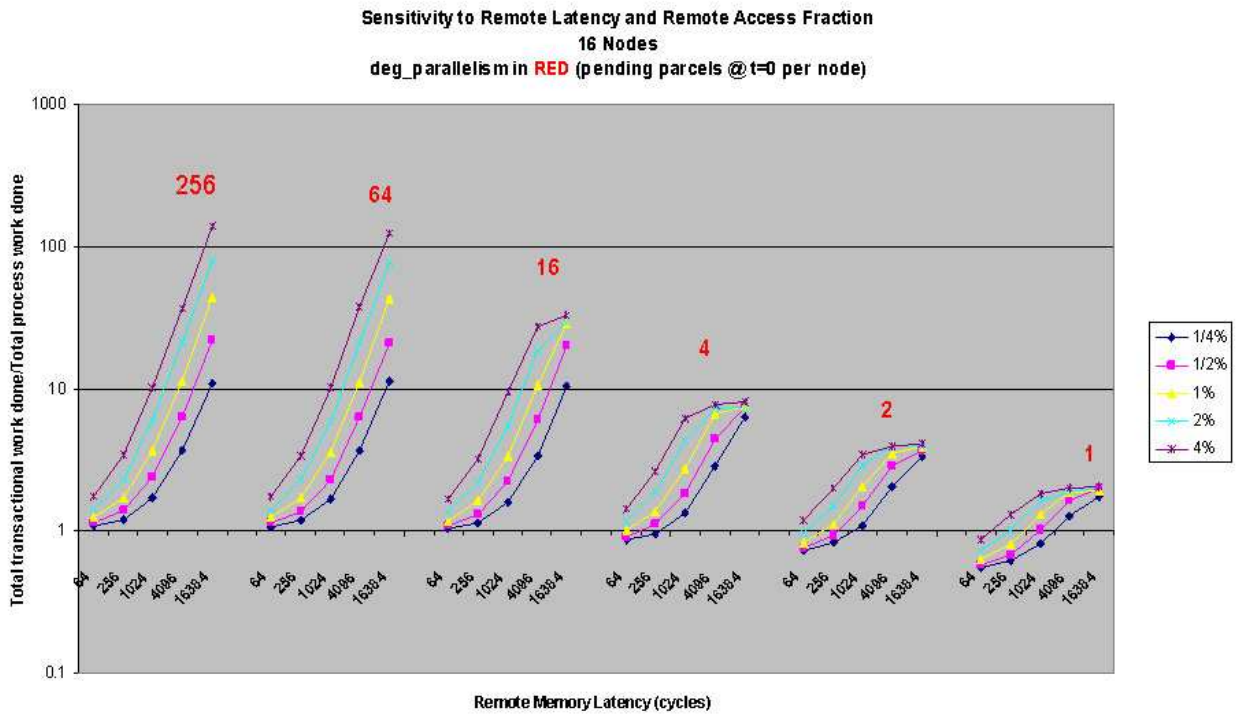


Figure 11: Latency Hiding with Parcels

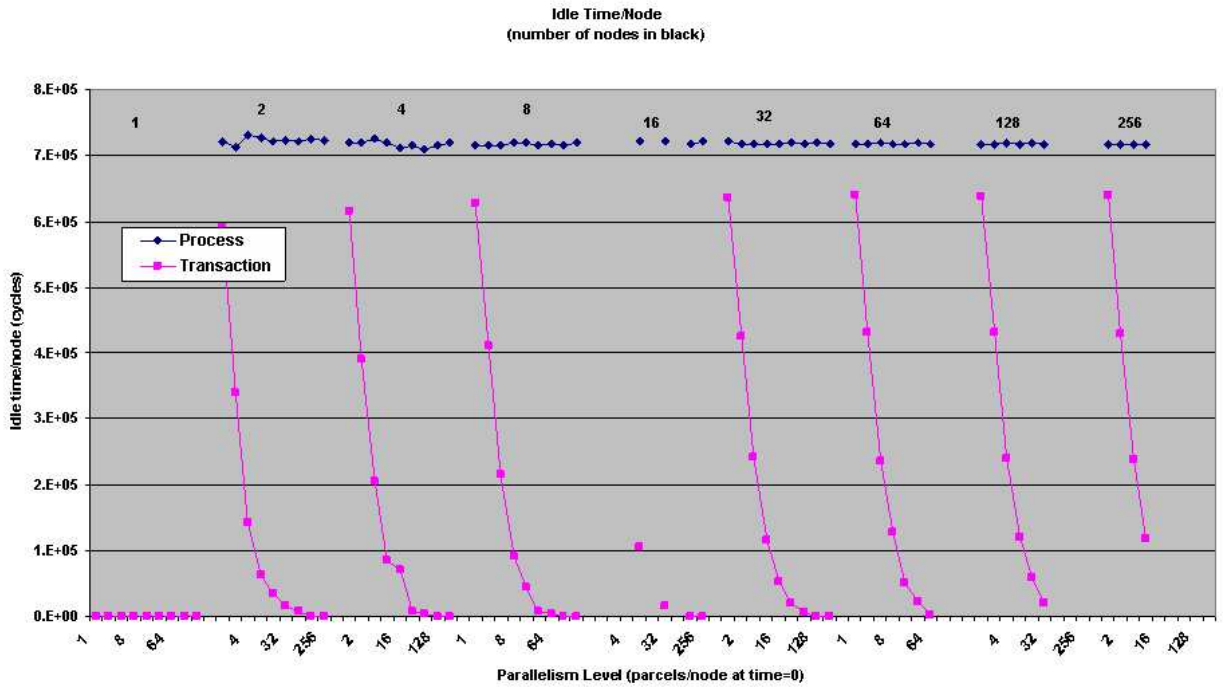


Figure 12: Idle Time with respect to Degree of Paralellism

system. This can be thought of as the average number of parcels per processors. For each major experiment, simulations are performed for different percentages of remote accesses (with respect to total number of memory accesses) as shown by the connecting lines. For each of these curves, the parameter measuring the number of cycles of system wide latency is varied. The vertical axis gives the ratio of number of operations performed by the test system with respect to those accomplished by the control system.

These results demonstrate that with sufficient parallelism and for systems with significant system-wide latency, the parcel split-transaction test systems perform much better than the control system, sometimes exceeding an order of magnitude in delivered performance. However, it also exposes certain operational regions where performance advantage is small or in fact reversed. This is particularly true when there is little parallelism and short system latencies.

Another way of examining the simulation results is the system idle times; i.e. the periods in which the processors are not performing operations but rather waiting for new work to do. This is shown in Figure 12. The actual values of the dependent parameter, the idle time, are not so important as they are as much an artifact of the arbitrary amount of time for which the experiment was carried out. Each major experimental set of simulations (there are 8 although some are less complete than others) differs in terms of the number of nodes in the two systems from single node systems on the left to systems comprising 256 nodes on the right. We didn't successfully complete the 16 node case. Each major experiment was performed for a number of cases varying in the degree of parallelism available for the test system. It is clear that for sufficient parallelism, the idle time drops virtually to zero for the test systems while the control system experiences relatively high idle time.

5 Discussion and Conclusions

We have developed a set of analytic and simulation models for exploring tradeoffs in the PIM design space. Two key questions were explored using the statistical parametric models. One was the potential performance advantage of systems incorporating PIM components. The second was the opportunity of supporting lightweight split-transaction operation of PIM nodes through parcel message-driven computation. In both cases, we found that there were major regimes of operation for which dramatic benefits could be realized. But that this was not true for all possible operating points and in certain cases performance degradation might be experienced.

5.1 Interrelationships between PIM Components and a Host Processor

Augmenting the memory system of a host processor with PIM components can yield performance gains ranging from moderate (a factor of 2 or less) to dramatic (an order of magnitude or more) for applications that can be separated into regions of high or low temporal locality. The models show that adding even small amounts of processing capability to the memory system can have significant impact. For data-intensive applications where there is little or no data reuse, and where caches are of little value, PIM may help enormously. The model that we developed for this study provides a strong foundation that characterizes the region of operation in terms of three independent variables: the number of PIM nodes, the fraction of work that can be assigned to PIM, and a third parameter that is both machine and application dependent. While it may be difficult to calibrate these parameters for specific design points, by sweeping them across a range, we are able to get a broad view of the design space and to recognize emerging trends.

In terms of ongoing research, this first study supports the direction and results by projects exploring PIM-enabled memory for conventional hosts, such as Diva [13] and Cascade.

5.2 Global Latency Hiding Using Parcels

This third and final study shows that parcels—based on and inspired by early work such as message-driven computation [9] [25] and active messages [32] [7] can have a dramatic effect in tolerating system-wide latency, but significant medium to fine-grain parallelism must be exposed in the applications to take advantage of this, and that efficient parcel handling mechanisms are required to realize performance gains. Further, as prior work has shown for MPPs [27], our model demonstrates that multithreading at the node can have tremendous benefit in PIM systems. Finally, execution models developed over a decade ago in the context of dataflow architectures such as Monsoon [26] and P-RISC [24] may have new relevance in PIM technology.

Acknowledgments

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The funding for this research was provided for by the Defense Advanced Research Projects Agency under task order number 15506, under the NASA prime contract number NAS7-1407.

References

- [1] N. R. Adiga and et. al. An overview of the bluegene/L supercomputer. In *Proceedings of Supercomputing (SC2002)*, Baltimore, MD, November 2002.
- [2] Arvind and Rishiyur S. Nikhil. Executing a program on the MIT tagged-token dataflow architecture. *IEEE Transactions on Computers*, 39(3):300–318, March 1990. Also appears in Proceedings of PARLE87. Parallel Architectures and Languages Europe.pp.1–29, vol.2.
- [3] J. Brockman, J. Zawodny, P. Kogge, and E. Johnson. Cache-in-Memory: A lower power alternative. Barcelona, Spain, June 1998. Workshop on Power-Driven Microarchitecture, held in conjunction with the International Symposium on Computer Architecture.
- [4] J. B. Brockman, E. Kang, S. Kuntz, and P. Kogge. The architecture and implementation of a microserver-on-a-chip. Technical Report CSE TR02-05, University of Notre Dame CSE Dept., 2002.
- [5] Jay B. Brockman, Peter M. Kogge, Vincent W. Freeh, Shannon K. Kuntz, and Thomas L. Sterling. Microservers: A new memory semantics for massively parallel computing. In *Conference Proceedings of the 1999 International Conference on Supercomputing*, pages 454–463, Rhodes, Greece, June 20–25, 1999. ACM SIGARCH.
- [6] Microsoft Corporation. www.microsoft.com.
- [7] David E. Culler, Seth Copen Goldstein, Klaus Erik Schauer, and Thorsten Von Eicken. TAM – A compiler controlled Threaded Abstract Machine. *Journal of Parallel and Distributed Computing*, 18(3):347–370, July 1993.
- [8] W. J. Dally. *The J-Machine System*. Artificial Intelligence at MIT: Expanding Frontiers. MIT Press, 1990.
- [9] W. J. Dally, A. Chien, J. A. S. Fiske, G. Fyler, W. Horwat, J. S. Keen, R. A. Lethin, M. Noakes, P. R. Nuth, and D. S. Wills. The message driven processor: An integrated multicomputer

- processing element. In *International Conference on Computer Design, VLSI in Computers and Processors*, pages 416–419, Los Alamitos, CA, October 1992. IEEE Computer Society Press.
- [10] William Dally, Andrew Chang, Andrew Chien, Stuart Fiske, Waldemar Horwat, John Keen, Richard Lethin, Michael Noakes, Peter Nuth, Ellen Spertus, Deborah Wallach, and D. Scott Wills. The J-machine: A retrospective.
- [11] Monty Denneau. Blue gene. In *SC2000: High Performance Networking and Computing*, pages 35–35, Dallas, TX, November 2000. ACM.
- [12] Richard Fromm, Stylianos Perissakis, Neal Cardwell, Christoforos Kozyrakis, Bruce McGaughy, David Patterson, Tom Anderson, and Katherine Yelick. The energy efficiency of IRAM architectures. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, volume 25,2 of *Computer Architecture News*, pages 327–337, New York, June 2–4 1997. ACM Press.
- [13] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Apoorv Srivastava, William Athas, Jay Brockman, Vincent Freeh, Joonseok Park, and Jaewook Shin. Mapping irregular applications to DIVA, A PIM-based data-intensive architecture. In *Supercomputing (SC'99)*, Portland, Oregon, nov 1999. ACM Press and IEEE Computer Society Press.
- [14] C. Hewitt and H. G. Baker. Actors and continuous functionals. Technical Report MIT-LCS-TR-194, MIT Laboratory of Computer Science, 1978.
- [15] W. Daniel Hillis. The connection machine. Technical Report AIM-646, Massachusetts Institute of Technology, September 1981.
- [16] Robert A. Iannucci. Toward a dataflow/von neumann hybrid architecture. In *Proc. 15th Annual Symposium on Computer Architecture, Computer Architecture News*, pages 131–140. ACM, May 1988. IBM/MIT.
- [17] HyPerformix Inc. www.hyperformix.com.
- [18] The MathWorks Inc. www.mathworks.com.
- [19] Graham Kirsch. Active memory device delivers massive parallelism. In *Microprocessor Forum*, San Jose, CA, October 2002.
- [20] P. M. Kogge. EXECUBE - A new architecture for scalable MPPs. In Dharma P. Agrawal, editor, *Proceedings of the 23rd International Conference on Parallel Processing. Volume 1: Architecture*, pages 77–84, Boca Raton, FL, USA, August 1994. CRC Press.
- [21] Christoforos Kozyrakis, Joseph Gebis, David Martin, Samuel Williams, Ioannis Mavroidis, Steven Pope, Darren Jones, and David Patterson. Vector IRAM: A media-enhanced vector processor with embedded DRAM. In IEEE, editor, *Hot Chips 12: Stanford University, Stanford, California, August 13–15, 2000*, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2000. IEEE Computer Society Press.
- [22] G. Lipovski and C. Yu. The dynamic associative access memory chip and its application to SIMD processing and full-text database retrieval. In *IEEE International Workshop on*

- Memory Technology, Design and Testing*, pages 24–33, San Jose, CA, August 1999. IEEE, IEEE Computer Society.
- [23] P. C. Messina, T. A. Sterling, and P. H. Smith. *Enabling Technologies for PetaFlops Computing*. MIT Press, 1995.
- [24] Rishiyur S. Nikhil and Arvind. Can dataflow subsume von Neumann computing? In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 262–272, June 1989.
- [25] M. D. Noakes, D. A. Wallach, and W. J. Dally. The J-machine multicomputer: An architectural evaluation. In Lubomir Bic, editor, *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 224–236, San Diego, CA, May 1993. IEEE Computer Society Press.
- [26] Gregory M. Papadopoulos and David E. Culler. Monsoon: An explicit token-store architecture. In *17th International Symposium on Computer Architecture*, number 18(2) in ACM SIGARCH Computer Architecture News, pages 82–91, Seattle, Washington, May 28–31, June 1990.
- [27] R. Saavedra-Barrera, D. Culler, and T. von Eicken. Analysis of multithreaded architectures for parallel computing. In *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, pages 169–178. ACM Press, 1990.
- [28] Ashley Saulsbury, Fong Pong, and Andreas Nowatzky. Missing the memory wall: The case for processor/memory integration. In *23rd Annual International Symposium on Computer Architecture (23rd ISCA'96)*, *Computer Architecture News*, pages 90–101. ACM SIGARCH, May 1996.
- [29] Burton Smith. A massively parallel shared memory computer. In ACM-SIGACT; ACM-SIGARCH, editor, *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 123–124, Hilton Head, SC, July 1991. ACM Press.
- [30] Burton J. Smith. A pipelined, shared resource MIMD computer. In *Proceedings of the 1978 International Conference on Parallel Processing*, pages 6–8, 1978.
- [31] Thomas Sterling and Larry Bergman. A design analysis of a hybrid technology multithreaded architecture for petaflops scale computation. In *Conference Proceedings of the 1999 International Conference on Supercomputing*, pages 286–293, Rhodes, Greece, June 20–25, 1999. ACM SIGARCH.
- [32] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauer. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 256–266, Gold Coast, Australia, May 1992.