

# Performance Modeling and Interpretive Simulation of PIM Architectures and Applications

Zachary K. Baker and Viktor K. Prasanna

University of Southern California, Los Angeles, CA USA  
zbaker@halcyon.usc.edu, prasanna@ganges.usc.edu  
<http://advisor.usc.edu>

**Abstract.** Processing-in-Memory systems that combine processing power and system memory chips present unique algorithmic challenges in the search for optimal system efficiency. This paper presents a tool which allows algorithm designers to quickly understand the performance of their application on a parameterized, highly configurable PIM system model. This tool is not a cycle-accurate simulator, which can take days to run, but a fast and flexible performance estimation tool. Some of the results from our performance analysis of 2-D FFT and biConjugate gradient are shown, and possible ways of using the tool to improve the effectiveness of PIM applications and architectures are given.

## 1 Introduction

The von Neumann bottleneck is a central problem in computer architecture today. Instructions and data must enter the processing core before execution can proceed, but memory and data bus speeds are many times slower than the data requirements of the processor. Processing-In-Memory (PIM) systems propose to solve this problem by achieving tremendous memory-processor bandwidth by combining processors and memory together on the same chip substrate. Notre Dame, USC ISI, Berkeley, IBM, and others are developing PIM systems and have presented papers demonstrating the performance and optimization of several benchmarks on their architectures. While excellent for design verification, the proprietary nature and the time required to run their simulators are the biggest detractors of their tools for application optimization. A cycle-accurate, architecture-specific simulator, requiring several hours to run, is not suitable for iterative development or experiments on novel ideas. We provide a simulator which will allow faster development cycles and a better understanding of how an application will port to other PIM architectures [4, 7]. For more details and further results, see [2].

<sup>1</sup> Supported by the US DARPA Data Intensive Systems Program under contract F33615-99-1-1483 monitored by Wright Patterson Airforce Base and in part by an equipment grant from Intel Corporation. The PIM Simulator is available for download at <http://advisor.usc.edu>

## 2 The Simulator

The simulator is a wrapper around a set of models. It is written in Perl, because the language’s powerful run-time interpreter allows us to easily define complex models. The simulator is modular; external libraries, visualization routines, or other simulators can be added as needed. The simulator is composed of various interacting components. The most important component is the data flow model, which keeps track of the application data as it flows through the host and the PIM nodes. We assume a host with a separate, large memory. Note that as the PIM nodes make up the main memory of the host system in some PIM implementations. The host can send and receive data in a unicast or multicast fashion, either over a bus or a non-contending, high-bandwidth, switched network. The bus is modeled as a single datapath with parameterized bus width, startup time and per element transmission time. Transmissions over the network are assumed to be scheduled by the application to handle potential collisions. The switched network is also modeled with the same parameters but with collisions defined as whenever any given node attempts to communicate with more than one other node(or host), except where multicast is allowed. Again, the application is responsible for managing the scheduling of data transmission. Communication can be modeled as a stream or as packets.

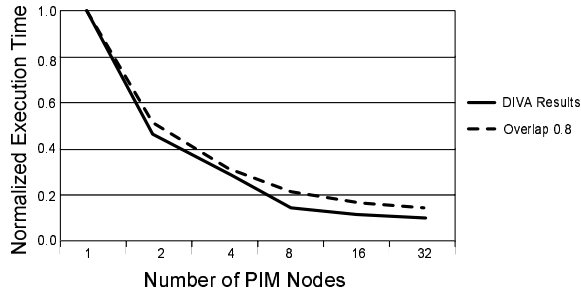
Computation time can be modeled at an algorithmic level, e.g.  $n \lg(n)$  based on application parameters, or in terms of basic arithmetic operations. The accuracy of the computation time is dependent entirely on the application model used. We assume that the simulator will be commonly used to model kernel operations such as benchmarks and stressmarks, where the computation is well understood, and can be distilled into a few expressions. This assumption allows us to avoid the more complex issues of the PIM processor design and focus more on the interactions of the system as a whole.

## 3 Performance Results

### 3.1 Conjugate Gradient Results

Figure 1 shows the overall speedup of the biConjugate Gradient stressmark with respect to the number of active PIM elements. It compares results produced by our tool using a DIVA parameterized architecture to the cycle-accurate simulation results in [4]. Time is normalized to a simulator standard. The label of our results, “Overlap 0.8”, denotes that 80% of the data transfer time is hidden underneath the computation time, via prefetching or other latency hiding techniques. The concept of overlap is discussed later in this paper.

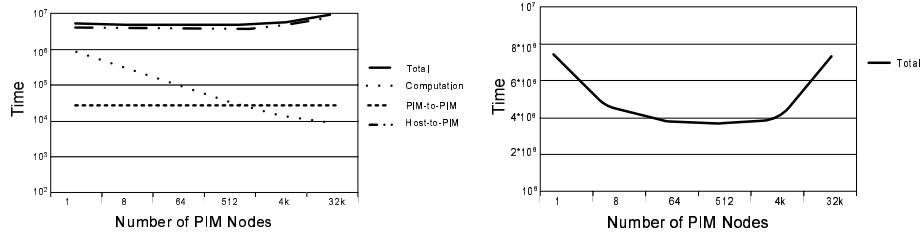
BiConjugate Gradient is a DARPA DIS stressmark [1]. It is used in matrix arithmetic to find the solution of  $y = Ax$ , given  $y$  and  $A$ . The complex matrices in question tend to be sparse, which makes the representation and manipulation of data significantly different than in regular data layout of FFT. The application model uses a compressed sparse row matrix representation of  $A$ , and load balances based on the number of elements filling a row. This assumes that the number of rows is significantly higher than the number of processors. All PIM nodes are sent the vector  $y$  and can thus execute on their sparse elements independently of the other PIM nodes.



**Fig. 1.** Speedup from one processor to  $n$  processors with DIVA model

Host-to-PIM transfer costs, computation time, and total execution time (total) as the number of PIM nodes increases under a DIVA model. The complete simulation required 0.21 seconds of user time on a Sun Ultra250 with 1024 MB of memory.

The graph shows that the computation time decreases linearly with the number of PIM nodes, and the data transfer time increases non-linearly. We see in the graph that PIM-to-PIM transfer time is constant—this is because the number of PIM nodes in the system does not dramatically affect the amount of data (a vector of size  $n$  in each iteration) sent by the BiCG model. Host-to-PIM communication increases logarithmically with number of PIM; the model is dependent mostly on initial setup of the matrices and final collection of the solution vectors. The Host-to-PIM communication increases toward the end as the communications setup time for each PIM becomes non-negligible compared to the total data transferred. Figure 2(right) shows a rescaled version of the total execution time for the same parameters. Here, the optimal number of PIM under the BiCG model and architectural parameters is clear—this particular application seems suited to a machine of 64 to 128 PIM nodes most optimally in this architecture model.

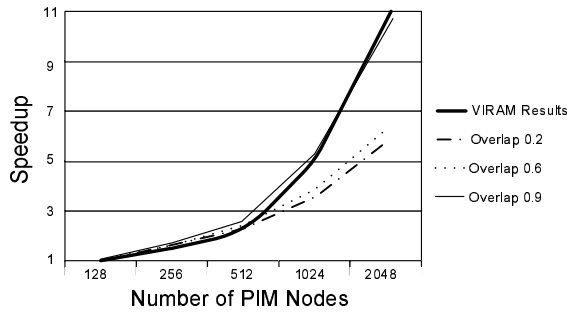


**Fig. 2.** BiConjugate Gradient Results; unit-less timings for various amounts of PIM nodes. (left: all results, right: total execution time only)

### 3.2 FFT

Another stressmark modeled is the 2-D FFT. Figure 3 shows execution time versus the number of FFT points for the Berkeley VIRAM architecture, comparing our results against their published simulation results [8]. This simulation,

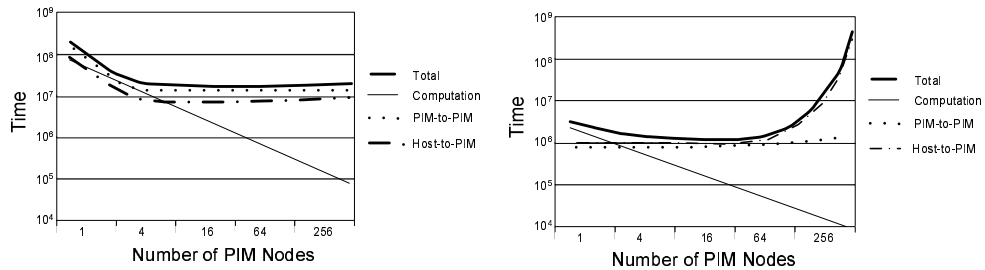
for all points, required 0.22 seconds of user time. The 2-D FFT is composed of a one dimensional FFT, a matrix transpose or ‘corner-turn’, and another FFT, preceded and followed by heavy communication with the host for setup and cleanup. Corner turn, which can be run independently of the FFT application, is a DARPA DIS stressmark [1]. Figure 3 shows the VIRAM speedup results against various overlap factors– a measure of how much of the data exchange can overlap with actual operations on the data. Prefetching and prediction are highly architecture dependent; thus the simulator provides a parameter for the user to specify the magnitude of these effects.



**Fig. 3.** Speedup versus number of FFT Points for various fetch overlaps, normalized to 128 points.

In the graph we see that the VIRAM results match most closely with an overlap of 0.9; that is, virtually all of the data transfer is hidden by overlapping with the computation time. This ‘overlap’ method is similar to the ‘clock multiplier factor N’ used by Rsim in that it depends on the application and the system and cannot to determined without experimentation [5].

Inspecting the VIRAM architecture documentation, we see that it includes a vector pipeline explicitly to hide the DRAM latency [6]. Thus our simulation results suggest the objective of the design has been achieved.



**Fig. 4.** 2-D FFT Results (left: Small memory size, right: Small problem size)

The simulator can be used to understand the performance of a PIM system under varying application parameters, and the architecture’s effect on optimizing those parameters. A graph of the simulator output in Figure 4(left) and 4(right) show a generic PIM system interconnected by a single wide bus. The

FFT problem size is  $2^{20}$  points, and the memory size of any individual node is 256K. The change in slope in Figure 4(left) occurs because the problem fits completely within the PIM memory after the number of nodes exceeds four. Until the problem size is below the node memory capacity, bandwidth is occupied by swapping blocks back and forth between the node and the host memory. Looking toward increasing numbers of PIM, we see that the total time has a minimum at 128, and then slowly starts to increase. Thus it could be concluded that an optimal amount of PIM nodes for an FFT of size  $2^{20}$  is 128.

## 4 Conclusions

In this paper we have presented a tool for high-level modeling of Processing-In-Memory systems and its uses in optimization and evaluation of algorithms and architectures. We have focused on the use of the tool for algorithm optimization, and in the process have given validation of the simulator's models of DIVA and VIRAM. We have given a sketch of the hardware abstraction, and some of the modeling choices made to provide an easier-to-use system. We have shown some of the application space we have modeled, and presented validation for those models against simulation data from real systems, namely DIVA from USC ISI and VIRAM from Berkeley.

This work is part of the Algorithms for Data IntensiVe Applications on Intelligent and Smart MemORies (ADVISOR) Project at USC [3]. In this project we focus on developing algorithmic design techniques for mapping applications to architectures. Through this we understand and create a framework for application developers to exploit features of advanced architectures to achieve high performance.

## References

1. Titan Systems Corporation Atlantic Aerospace Division. DIS Stressmark Suite. <http://www.aaec.com/projectweb/dis/>, 2000.
2. Z. Baker and V.K. Prasanna. Technical report: Performance Modeling and Interpretive Simulation of PIM Architectures and Applications. In preparation.
3. V.K. Prasanna et al. ADVISOR project website. <http://advisor.usc.edu>.
4. M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, A. Srivastava, W. Athas, J. Brockman, V. Freeh, J. Park, and J. Shin. Mapping Irregular Applications to DIVA, a PIM-based Data-Intensive Architecture. In *SC99*.
5. C.J. Hughes, V.S. Pai, P. Ranganathan, and S.V. Adve. Rsim: Simulating Shared-Memory Multiprocessors with ILP Processors, Feb 2002.
6. Christoforos Kozyrakis. A Media-Enhanced Vector Architecture for Embedded Memory Systems Technical Report UCB//CSD-99- 1059, July 1999.
7. D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM: IRAM, 1997.
8. Randi Thomas. An Architectural Performance Study of the Fast Fourier Transform on Vector IRAM. Master's thesis, University of California, Berkeley, 2000.