A New Perspective on Processing-in-memory Architecture Design

Dong Ping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph Greathouse, Mitesh Meswani
Mark Nutter, Mike Ignatowski

Research Group, AMD, California, USA
dongping.zhang@amd.com

Abstract

As computation becomes increasingly limited by data movement and energy consumption, exploiting locality throughout the memory hierarchy becomes critical for maintaining the performance scaling that many have come to expect from the computing industry. Moving computation closer to main memory presents an opportunity to reduce the overheads associated with data movement. We explore the potential of using 3D die stacking to move memory-intensive computations closer to memory. This approach to processing-in-memory addresses some drawbacks of prior research on in-memory computing and appears commercially viable in the foreseeable future. We show promising early results from this approach and identify areas that are in need of research to unlock its full potential.

Keywords processing-in-memory, memory system, data locality, performance model, memory-intensive applications

1. Introduction

While the advancement of processor technology has rapidly increased computational capabilities, improvements in bandwidth and latency to off-chip memory have not kept up. Further, an increasing proportion of power in computing systems is being spent on data movement, especially off-chip memory accesses. These problems are exacerbated for emerging workloads that exhibit memory intensive behaviors with irregular access patterns and limited data reuse. Moving computation closer to where the data reside has the potential to improve both the performance bottlenecks and energy costs associated with memory access.

We investigate the benefits of reducing the aggregate distance of data movement, and hence its associated energy consumption, through processing-in-memory (PIM) implemented using 3D die stacking. There are two aspects to achieving this: firstly, to plan the data layout in memory such that in-memory computation with a high degree of locality becomes feasible; secondly, to dispatch computation to locations where the input data reside. Programming models and systems software also play a key role in enabling the realization of PIM capabilities. Therefore we highlight the challenges that must be addressed by architecture and software communities to fully exploit the potential of in-memory processing.

2. Background

PIM attracted significant attention in the research community for a short period around the beginning of this century [1, 2]. Many of those efforts focused on one of two approaches. Efforts such as IRAM [3] integrated embedded DRAM on logic chips. However, this approach could not cost-effectively accommodate sufficient memory capacity for mainstream high-performance systems due to the reduced density of embedded memory. Efforts such as ActivePages [4], DIVA [5] and FlexRAM [6] integrated logic on memory dies. However, due to the reduced performance of logic implemented in DRAM processes (typically multiple process generations behind contemporary logic processes), such approaches resulted in in-memory processors with reduced performance or highly specialized architectures geared only for select operations. This in turn limited the applicability and programmability of such PIMs which, along with the cost implications of reduced DRAM density due to the presence of compute logic, limited the adoption of such approaches.

We revisit the PIM concepts utilizing 3D die stacking to tightly couple processors implemented in a logic process with memories implemented in memory processes using through-silicon via (TSV). This approach presents a different tradeoff in the PIM design space that has somewhat lower bandwidth than processors on memory dies but circumvents the issues of logic performance and memory density. Further, this approach potentially enables the use of commercially viable memory dies. A similar approach has been advocated in a recent FlexRAM retrospective paper by J. Torrellas [7].

Direct stacking of memory on a processor limits memory capacity to what fits within the processor die's footprint. Further, processor performance is throttled due to thermal considerations. To circumvent these limitations, we explore the integration of PIM within 3D memory stacks as auxiliary processors under the control of a standalone host processor. While prior research has explored issues in 3D stacking of processors and memory [8, 9], our focus is on evaluating the performance characteristics of a compute node consisting of a mainstream processor with a number of PIM-enabled memory stacks attached to it. We focus our initial explorations on high-performance computing (HPC) and other such domains that require large memory capacity as well as high compute throughput.

3. PIM for HPC

An example node configuration suitable for our initial focus is shown in Figure 1 from a high-level point of view. An overall system geared for HPC may consist of many instances of the same or similarly configured nodes. Here, an individual node includes a high-performance host processor as well as stacked DRAM with PIM.

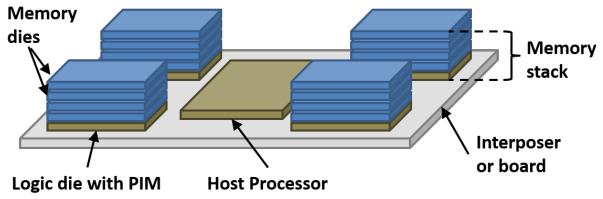

Figure 1. An example compute node design with PIM

The example use cases and respective benefits from PIM include, but are not limited to: replacing read-update-store with op-and-store to reduce the round-trips to memory and avoid synchronization; increasing the efficiency of indirect memory accesses, e.g., pointer chasing, arbitrary gather/scatter; avoiding reading data to the host

MSPC'13, June, 2013, Seattle, Washington.

during memory layout transformation operations, e.g., matrix transpose, convolutions; executing predefined memory intensive operations, e.g., reductions; last, and more broadly, executing arbitrary program consisting of user-defined memory-intensive calculations.

PIM Simulation and Performance Model

Two broad categories of performance modeling are commonly studied in the literature: structural and analytical models [10]. The former uses simulation techniques to model systems, while the latter abstracts design factors of interest into an analytical expression to predict performance. Cycle-accurate simulation is commonly used to evaluate design proposals because it allows researchers to directly model how architectural changes interact with the system. However, these simulators typically run many orders of magnitude slower than native execution and quickly become intractable for workloads with large data sets and unstructured access patterns, which are of high interest for PIM evaluations. Therefore, we rely on analytical models in conjunction with statistics gathered during native execution on today's hardware for early exploration of the PIM design space.

We design performance models to estimate the effects of changes to PIM configuration, stacked DRAM, and interconnection between these components. In essence, we run applications annotated with PIM extensions on existing hardrware and gather both hardware-supplied performance-monitoring data and the program's critical paths through happens-before analysis. A performance scaling model is built based on the statistics collected, information about the underlying hardware and the modeled PIM system.

Evaluation

As a preliminary study, we evaluated the presented PIM architecture over two simple test cases: a multi-threaded implementation of prefix sum computation and a data-parallel implementation of a weighted sum of vectors kernel. The former is primarily latency bound while the latter is bandwidth-limited.

Table 1 shows the performance of the parallel prefix sum benchmark on four CPU PIM processors normalized to the execution on a 4-core host processor. Each PIM processor in config. 1 and 2 consists of one CPU with normalized attributes compared with host CPU listed in Table 1. As expected, performance improves for configurations with lower memory latency despite the reduced execution frequency of PIM processors. While the specific latency reduction in a PIM processor is implementation-dependent, factors such as the elimination of off-chip transfers, shorter wire delays and simpler cache hierarchies can contribute to this reduction.

Table 1: Normalized parallel prefix sum performance on PIM

	Baseline	PIM config. 1	PIM config. 2
Host CPU cores used	4		
PIM cores used		4	4
Normalized core freq.	1	0.5	0.5
Normalized memory latency	1	0.7	0.5
Normalized execution time	1	0.87	0.77

Table 2: Normalized scaled vector add kernel performance on PIM

	Baseline	PIM config. 3	PIM config. 4
Normalized FLOPs used	1	0.27	0.27
Normalized memory bandwidth	1	2	4
Normalized execution time	1	0.51	0.26

Table 2 shows the performance of the weighted sum of vectors kernel running on 4 GPU PIM processors normalized to the execution on a high-performance GPU. Each PIM processor in config. 3 and 4 consist of one GPU with relative attributes normalized to the host GPU configuration as shown in Table 2. To ensure a fair comparison, we assume the baseline architecture also uses stacked memory mounted on an interposer with the processor to achieve high bandwidth. The PIM processors are still able to achieve higher intra-stack bandwidth. We make conservative assumptions about intra-stack bandwidth that could still enable the use of commodity DRAM dies in a PIM-enabled memory stack. Modifications to the DRAM dies could enable even greater performance gains.

These examples are prelimilary tests to demonstrate the potential benefits of PIM architecture design for two types of applications: latency-bound and bandwidth-bound. Further application studies are essential to better understand potential usage models and benefits of PIM as well as to drive further development and refinement of our evaluation tools and methodology. While the above evaluation focuses on performance, this approach also enables significant energy savings due to a variety of factors including reduced off-chip and on-chip data movement and execution of memory-bound tasks on in-memory processors that are less aggressively designed (relative to host processors).

Software Ecosystem

There are various issues to be considered around the interaction between system software and PIM. One potential design point is to assume that PIM devices are fully capable processors that can share a single virtual address space with the host. This enables a path for a range of multi-core OS capabilities to be adapted to in-memory processors including virtual memory, preemptive multi-tasking, and placement of tasks and memory objects in an environment supporting non-uniform memory access. The application programming model can potentially resemble existing models for heterogeneous multi-core architectures. These include standard CPU threading and tasking models exposed via PIM-augmented APIs or additionally via extensions to pragmas such as OpenMP. Evolutions of standards for heterogeneous compute, such as OpenCL, may additionally enable task- and data-parallel programming models. Under this scenario, legacy applications can be adapted to PIM and execute correctly with only minimal modifications. Higher-level abstractions such as libraries and domain-specific languages can ease the development of highly-tuned applications for PIM-enabled systems. Run-time systems that exploit compile-time and dynamic profiling information may optimize task placement between host and in-memory processors to further ease the transition to PIM-enabled architectures.

4. Conclusion

As computation becomes increasingly limited by data movement and energy consumption, exploiting locality throughout the memory hierarchy becomes critical to maintaining the performance scaling that many have come to expect from the computing industry. Emerging 3D die stacking technology provides an opportunity to exploit in-memory computation in a practical manner. Our early evaluations show promising results in the ability of this technique to address data movement bottlenecks and overheads.

However, broad adoption of PIM can only occur if there are appropriate programming models and languages along with adequate compiler, runtime, and operating system support. Simultaneous advancement in these aspects are yet to be addressed. While HPC and other memory-intensive applications make a good starting point and are likely to be among the first beneficiaries of PIM-enabled systems, nearly every segment of the computing landscape can benefit from the performance and energy improvements that can be achieved through PIM. We also need to take the cost of application refactoring into consideration while addressing the aforementioned subjects. Therefore, high-level abstractions that express programmer intent and available locality in architecture independent forms are also important as they can enable efficient mapping to novel architectures such as PIM with reduced programmer effort.

These areas present a number of exciting opportunities that the research community must address to realize the full potential of PIM capabilities that are enabled by emerging implementation techniques.

References

[1] D. Elliott et al. Computational RAM: Implementing Processors in Memory. IEEE Design & Test, Volume 16 Issue 1, 1999.

[2] J. Lee et al. Automatically mapping code in an intelligent memory architecture. International Symposium on HPCA, 2001.

[3] B. R. Gaeke et al. Memory-Intensive Benchmarks: IRAM vs. Cache-Based Machines. IPDPS, 2002.

[4] M. Oskin et al. Active pages: a computation model for intelligent memory. International symposium on computer architecture, 1998

[5] J. Draper et al. The architecture of the DIVA processing-in-memory chip. Interntional Conference on Supercomputing, 2002.

[6] Y. Kang et al. FlexRAM: toward an advanced intelligent memory system. International Conference on Computer Design, 1999.

[7] J. Torrellas. FlexRAM: Toward an Advanced Intelligent Memory System. A Retrospective Paper, ICCD, Sep. 2012

[8] G. H. Loh, 3D-Stacked Memory Architectures for Multi-core Processors. ISCA, 2008.

[9] C. Liu et al. Bridging the processor-memory performance gap with 3D IC technology. IEEE Design & Test of Computers, 2005.

[10] S. Pllana et al. Performance Modeling and Prediction of Parallel and Distributed Computing Systems: A Survey of the State of the Art. International Conference on CISIS, 2007