TOP-PIM: Throughput-Oriented Programmable Processing in Memory

Dong Ping Zhang[1] Nuwan Jayasena[1] Alexander Lyashevsky[1] Joseph L. Greathouse[1]
Lifan Xu[2*] Michael Ignatowski[1]

[1]AMD Research
{Dongping.Zhang, Nuwan.Jayasena, Alexander.Lyashevsky, Joseph.Greathouse, Mike.Ignatowski}@amd.com

[2]Dept. of Computer and Information Sciences
University of Delaware
xulifan@udel.edu

ABSTRACT

As computation becomes increasingly limited by data movement and energy consumption, exploiting locality throughout the memory hierarchy becomes critical to continued performance scaling. Moving computation closer to memory presents an opportunity to reduce both energy and data movement overheads. We explore the use of 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 is commercially viable in the foreseeable future.

Because 3D stacking provides increased bandwidth, we study throughput-oriented computing using programmable GPU compute units across a broad range of benchmarks, including graph and HPC applications. We also introduce a methodology for rapid design space exploration by analytically predicting performance and energy of in-memory processors based on metrics obtained from execution on today's GPU hardware. Our results show that, on average, viable PIM configurations show moderate performance losses (27%) in return for significant energy efficiency improvements (76% reduction in EDP) relative to a representative mainstream GPU at 22nm technology. At 16nm technology, on average, viable PIM configurations are performance competitive with a representative mainstream GPU (7% speedup) and provide even greater energy efficiency improvements (85% reduction in EDP).

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles—*heterogeneous (hybrid) systems*

Keywords

Processing in memory, performance modeling and analysis, energy efficiency, GPGPU

*Note: Lifan Xu contributed to the application studies during his internship with AMD Research.

HPDC'14, June 23–27, Vancouver, BC, Canada.

ACM 978-1-4503-2749-7/14/06 ...$15.00.

1. INTRODUCTION

Processors have steadily become more computationally capable and energy efficient, but improvements in bandwidth, latency and energy consumption of off-chip memory accesses have not kept pace with advances in processor architectures [36, 40]. As a result, the memory system is often a performance bottleneck and accounts for a significant, and increasing, fraction of system level energy consumption [30, 43]. A 64b DRAM access now consumes nearly two orders of magnitude more energy than a double-precision floating point arithmetic operation [4, 15, 27].

Memory system energy consumption is of particular importance for future high-performance computing systems. Sample system goals for the US Department of Energy Exascale efforts include a memory bandwidth of 4 TB/s per node at a system size of 100,000 nodes within a 20 MW power budget [46]. Even with aggressive assumptions about memory and interface technology improvements reducing total DRAM access energy from approximately 60-80 pJ/b for DDR3 [4, 15] to 4 pJ/b[1] [27, 44], sustaining 4 TB/s per node over 100,000 nodes will consume 70% of the entire system's power budget on DRAM accesses alone.

This paper explores the potential of processing in memory (PIM) implemented via 3D die stacking to reduce memory access energy and improve performance. Recent industry trends suggest the imminent adoption of 3D die stacking in mass-produced memory parts [11]. Some vendors have developed DDR3 devices that internally incorporate 3D stacking to increase capacity [5]. Multiple memory vendors are participating in the Hybrid Memory Cube (HMC) consortium aimed at commercializing "memory cubes" consisting of 3D stacked DRAM dies atop a "base" logic die [37]. The Wide I/O JEDEC standard for stacking memory with logic devices aimed at mobile applications was released in early 2012 [1]. A similar JEDEC standard for high-performance applications, High Bandwidth Memory (HBM), was released recently [2]. A number of academic publications have also explored the stacking of DRAM on logic dies [31, 35, 47].

Thermal challenges are a key impediment to stacking memory directly on top of a high-performance processor. Heat generated by the processor reduces the retention time of data in DRAM, requiring the throttling of processor performance and/or increasing memory refresh rate, neither of which is desirable in high-performance systems. In this paper, we explore a system organization where memory is not

[1]Note that some predictions are much less optimistic (*e.g.*, 25 pJ/b in 7nm technology [15]).

stacked directly on the main compute processor. Instead, an auxiliary, in-memory processor is incorporated on the base logic die of each memory stack as shown in Figure 1. Memory-intensive code may be offloaded to these in-memory processors to exploit the high bandwidth and low-energy to memory enabled by being stacked directly under the memory. As these in-memory processors are geared towards running memory-intensive code, their compute resources can be optimized for low-energy operation and reduced thermals. As the main compute processor ("host") does not have memory stacked on it, it is not subject to stringent thermal constraints and can support high performance for compute-intensive code. The primary goal of this study is to determine the performance and energy characteristics of such auxiliary, in-memory processors across a wide range of applications.

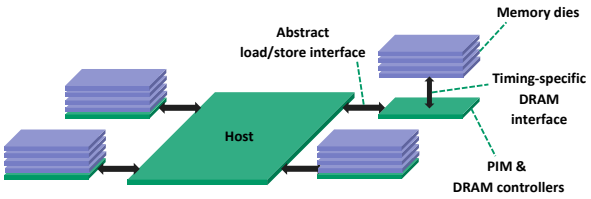

Figure 1: An example system with in-memory processors

Prior research has shown that the most significant performance benefit of stacked DRAM is increased bandwidth [16]. This motivates the incorporation of data-parallel accelerators in the in-memory processors to effectively utilize the available bandwidth. In this study, we focus on GPGPU execution units as the in-memory data-parallel accelerators. The programmability of modern GPUs also enables GPU-accelerated in-memory processors to be utilized over a wide range of applications. Further, the energy efficiency of GPU architectures (in terms of operations/W) also helps reduce thermal concerns for a given level of performance.

Evaluating PIM also presents new simulation challenges. Applications that can best exploit the benefits of PIM are those with large data sets that do not effectively fit into caches. Furthermore, the design space of PIM systems is larger than traditional designs, as both PIM and host configurations must be explored. Slow microarchitectural simulators hinder the ability to execute the necessary applications in a reasonable amount of time, which limits the state space that can be studied. As a result, a fast simulation methodology is crucial for exploring a sufficiently broad spectrum of applications and relevant design points. In order to address this challenge, we propose a methodology that first gathers hardware performance and power statistics during execution on current hardware. This data is then fed into a machine learning (ML) model that predicts the performance and power on future PIM and host hardware configurations.

This paper makes the following primary contributions:

- We explore the viability of GPU-accelerated architectures as in-memory processors and explore their system design space in near-future technology nodes. To our knowledge, this is the first study of using GPUs for in-memory computing in order to accelerate a non-stacked host processor.
- In order to enable rapid design space exploration, we present a simulation methodology that automatically scales performance and power values from existing hardware to future design points and quantify its accuracy.
- We evaluate the energy and performance impact of PIM in near-future technology nodes across a broad range of GPGPU workloads and identify characteristics that make a workload amenable to offload to a throughput-oriented, in-memory accelerator.

The remainder of the paper is organized as follows. Section 2 provides relevant background information. Section 3 describes our proposed system organization with GPU-accelerated in-memory processors. Section 4 describes and characterizes the benchmarks used for our evaluations. Sections 5 and 6 present the performance and power models we have developed for exploring PIM organizations. Section 7 presents and discusses evaluation results. Sections 8 and 9 discuss future directions and related prior work. Finally, Section 10 summarizes our findings and concludes.

2. BACKGROUND AND MOTIVATION

2.1 3D Die Stacking

Vertical stacking of logic and memory dies has been widely discussed in the research literature [12, 28, 31, 32]. We focus on two primary high-level architectural implications of 3D stacking. First, 3D stacking allows multiple implementation technologies to be integrated within a stack, allowing DRAM and logic dies to be coupled together. This is fundamental to our in-memory architecture, in contrast to previous works that try to implement both computation and storage in the same design process. Second, interconnections within a die stack, in the form of through silicon vias (TSV), enable higher bandwidth, lower latency and lower energy communication among the dies within a stack, relative to 2D organizations. The improved bandwidth arises primarily from higher TSV density and the ability to clock these "on-chip" links at higher frequencies than off-chip links with reasonable complexity and energy overheads. TSV pitches of 10-50μm were reported as of 2011 across a variety of vendors, and ITRS roadmaps predict 4-8μm global TSV pitches in the 2015-2018 time frame [7]. Latency and energy benefits arise from the shorter, on-chip vertical distances traversed, and the reduced capacitance compared to off-chip connections and longer wires on 2D organizations. Prior work has estimated that traversal latency of even an extreme case of 20 dies stacked vertically to be on the order of 12ps [32].

2.2 Memory Power

The energy impact of off-chip memory is amplified by the characteristics of today's mainstream DRAM interfaces. Delay/Phase Locked Loops (DLL/PLL) and clocks in high-performance DRAM interfaces such as DDR3 and GDDR5 consume significant energy even when no data is being transferred resulting in high idle power consumption and poor energy proportionality. Prior studies have shown that the effective energy per bit for DDR3 increases by integer factors at low utilization due to interface overheads [33]. Some estimates suggest that 70% or more of the energy per DRAM access is consumed in the DDR3 interface [27, 29]. While mobile DRAM standards (*e.g.*, LPDDR2, LPDDR3) provide lower idle power consumption, they do so at the cost

of reduced bandwidth. Emerging memory standards based on high-speed serial links, such as Hybrid Memory Cube (HMC) [37], introduce other interface overheads, including those due to the energy and latency of serialization and de-serialization. The "short reach" SERDES physical interface (PHY) of HMC is expected to consume 5-10 pJ/b out of an expected memory energy budget of 13-20 pJ/b [10, 37]. On the other hand, traversal of a 3D TSV is expected to consume on the order of 30-110 fJ/b [4]. Therefore, there is significant potential to reduce effective memory energy consumption by incorporating die-stacked, in-memory processors and reducing the overheads associated with high-bandwidth, off-chip interfaces.

2.3 GPU Architecture

At a high level, GPU architectures consist of collections of simple execution units (ALUs) operating under a single-instruction, multiple-thread execution model. We base the discussion in this paper on AMD's Graphics Core Next (GCN) GPU core architecture [8]. A simplified overview of a GCN-like GPU is shown in Figure 2. The computing resources are grouped into Compute Units (CU). From a GPGPU perspective, each CU is composed of four 16-wide SIMD units (along with associated register resources), a scalar unit, L1 cache and a shared scratchpad. The off-chip memory (DRAM) is organized as a set of memory channels, each with an associated slice of the L2 cache. The physical address space is striped among the memory channels and the CUs access L2 and DRAM via an on-chip network.

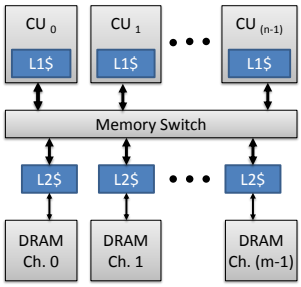

Figure 2: Simplified GPGPU architecture overview

GPUs are highly multi-threaded, with each CU simultaneously running hundreds of threads. This helps them tolerate high-latency operations, such as memory delays; when a thread is stalled due to a long-latency operation, other threads can utilize the available hardware resources. This, in turn, maximizes the computational and memory throughput of GPUs without requiring complex, power-hungry hardware such as out-of-order execution. As such, GPUs can effectively utilize vast amounts of memory bandwidth (consumer GPUs often have 10× or more bandwidth than consumer CPUs). This ability motivates our desire to put GPUs in high-bandwidth in-memory configurations.

2.4 Power and Thermal Considerations

PIM designs raise a number of power-related questions. Like other die-stacked designs [12], PIM can run into thermal constraints because in-memory processor and memory share the same path to the heatsink. Heat from the in-memory processor can raise the temperature of the DRAM, which places tight bounds on the amount of power PIM can use. The normal operating temperature range for DRAM is considered to be under 85 °C. Any increase in temperature beyond that requires refresh rates to be increased (typically doubled). As such, power estimation is an important aspect of studying PIM and we will factor this consideration into our architecture choices as described in Section 3.2.

3. THROUGHPUT-ORIENTED PIM

In this section, we qualitatively discuss stacked memory organization options and present our architecture with GPU-accelerated in-memory processors.

3.1 Stacked Memory Organization

Anticipated usage scenarios of Wide I/O and HBM, as well as academic studies of 3D stacked memory, predominantly assume one of two organizations: (1) direct 3D stacking of memory on a processor or (2) "2.5D" stacking where processor die(s) and memory stack(s) are mounted side-by-side on a silicon interposer. The first organization provides the benefit of tight coupling between processor and memory but raises thermal challenges. As a result, the thermal envelope of the processor must be constrained to avoid excessively degrading retention time of the stacked DRAM, reducing peak compute performance. The refresh rate of the DRAM may also need to be increased due to heat from the processor, further reducing performance. These constraints can offset a significant fraction of the performance potential of die-stacked memories, especially in high-performance systems. Further, the memory capacity that can be stacked in this organization is limited by the footprint of the processor.

The second organization (2.5D) reduces thermal concerns but incurs the additional cost of an interposer and introduces energy and latency overheads due to all DRAM accesses traversing the interposer. Bandwidth through the interposer may also be lower than that of 3D stacking because it is difficult to have as many wires in the interposer as there are TSVs in the 3D stack, necessitating a reduction in parallel communication channels and bandwidth.

A third alternative, adopted in HMC, is to integrate only the memory controller(s) and other memory support logic on the base dies of the memory stacks. The main processor is not stacked with memory and communicates with memory stacks via board-level or in-package links. This approach avoids thermal and capacity limitations but falls short of the performance and energy benefits of 3D or 2.5D stacking for bandwidth-intensive applications.

The approach we consider, shown in Figure 1, combines desirable aspects of the above three organizations. In-memory processors incorporated on the base logic dies of memory stacks provide the full bandwidth and energy efficiency of true 3D stacking for executing memory-intensive application segments. This alleviates the bandwidth demands on the links between host and memory, enabling board- and package-level interconnects for those links, unlike the more expensive interposer-based solutions required for 2.5D organizations. The host processor does not have stacked memory, thereby avoiding stringent thermal constraints, and can support high performance for compute-intensive code. A similar approach was proposed by Balaprakash *et al.* in an application-centric study [9]. In this work, we further incorporate realistic hardware area and power constraints. In addition, we analyze a specific processor microarchitecture

(GPU) to determine practical SoC design points and analyze other system bottlenecks beyond memory bandwidth.

3.2 Processor Architecture

Both the host and in-memory processors in our system organization are accelerated processing units (APU). Each APU consists of CPU and GPU cores on the same silicon die. We believe the choice of an APU as the in-memory processor has several benefits. First, the CPU and GPU components support familiar programming models and lower the barrier of using in-memory processors. Second, the programmability allows a broad range of applications to exploit PIM. Third, the use of existing GPU and CPU designs lowers the investment and risk inherent in developing PIM. Finally, the architectural uniformity of the host and in-memory processors ease porting of code to exploit PIM. Porting a subset of an application's routines (*e.g.*, the memory-intensive kernels) to PIM does not require a fundamental rewrite as the same languages, run-time systems and abstractions are supported on both host and in-memory processors. Syntactically, simply annotating the routines that are to be executed using PIM is sufficient. Due to these reasons, we adopt an existing GPU core microarchitecture for PIM. However, to our knowledge, this is the first study to consider a system organization incorporating GPU-accelerated architectures for in-memory computing.

The key programmability challenge in porting applications to our system organization is the distribution of data among multiple memory stacks and ensuring the locality of PIM computation with the associated data. For the purposes of this study, we designed a low-level API that allows programmer control over what data is allocated in a given stack and the ability to dispatch the associated computation to that stack. Further, we study throughput-oriented computations that are natural candidates for effectively utilizing the increased memory bandwidth available to in-memory processors. Consequently, we primarily focus on the GPU execution engines here.

We consider two potential PIM design points geared towards high-performance applications, one in 22nm technology and another in 16nm technology. The relevant hardware configuration parameters are listed in Table 1. The characteristics of these design points were determined based on past GPU architecture scaling trends and publicly available projections of future implementation technologies [7, 14]. As we focus this evaluation on GPU compute aspects, we omit the CPU configuration parameters. We do not include application code execution on the CPU cores of either host or PIM APUs in our evaluations.

As has been the recent trend, we assume host GPU architecture scaling occurs primarily through increasing the number of CUs and not through higher frequencies. We approximate this by holding the CU frequency roughly comparable to today's high-end discrete GPUs over the technology generations studied. The number of host CUs were selected to provide reasonable approximations of possible high-performance APU design points in future technologies. We made no effort to normalize host CU resources across the two configurations (*e.g.*, the host of the 16nm configuration incurs a 33% increase in area and a 34% increase in thermal design power (TDP) attributable to CUs over the 22nm configuration due to doubling of the number of CUs).

Table 1. Host and PIM configurations

	Config. 1	Config. 2
Technology node (nm)	22	16
Host		
Number of CU	32	64
Freq (MHz)	1000	1000
DRAM BW (GB/s)	320	640
DRAM BW/stack (GB/s)	160	160
DRAM capacity per stack (GB)	2	4
Number of DRAM stacks	2	4
Per-stack PIM		
Number of CU	8	12
Freq (MHz)	650	650
DRAM BW per stack (GB/s)	640	640

In-memory processor organizations of the two configurations were selected to fit within the constraints of a DRAM stack. Historically, DRAM die sizes have hovered in the range of 40-80 mm^2 [44]. We assume a logic base die size near the upper end of that range and the CU count of each configuration is selected to not exceed 50% of the logic die's area (including CU support hardware structures, shared caches etc.). The configurations of the in-memory processors were also constrained to not exceed 10W TDP attributable to CUs, caches and support hardware. According to our thermal models, this power level leaves sufficient headroom for other components within the stack while not exceeding 85 °C (and thereby avoiding the need to increase DRAM refresh rates) with commercially viable air-cooling solutions for high-performance systems. Note that these assumptions are more conservative, and result in lower power density, than some previous studies incorporating stacked DRAM [25]. We have also selected conservative operating frequencies for the in-memory processors to reduce dynamic and static power consumption.

We assume the internal CU microarchitecture (including cache hierarchy) does not change significantly from today's high-end GPUs over the period studied as well as between host and PIM implementations. Naturally, this is an unrealistic assumption from a microarchitectural perspective as CU implementations will continue to be refined and improved and variations in cache hierarchy may be desired between host and in-memory processors. However, our focus here is to understand the high-order performance and energy effects of in-memory computation relative to traditional organizations. Therefore, we assume that improvements internal to the processor are likely to benefit both PIM and host implementations in corresponding degrees and defer the evaluation of the impact of microarchitectural evolution to future work.

3.3 Memory Organization

In keeping with recent trends, we assume a memory bandwidth to compute ratio slightly greater than today's high-end GPUs at the 22nm node. While we double the raw bandwidth at the 16nm point, we hold the ratio of bandwidth to compute constant. We loosely model the memory interfaces from host to memory stacks on publicly available data on HMC [3, 37]. Each memory stack provides 160GB/s of bandwidth to the host processor. For PIM configurations, we assume intra-stack memory bandwidth four times greater

than externally available bandwidth. We explore the sensitivity to this factor in Section 7.

Figure 1 also identifies the key memory interfaces in our organization. As requests from both the host and PIM must be serviced by the DRAM, the DRAM controllers reside on the logic dies in the memory stacks. Therefore, the interface subject to DRAM timing constraints is the vertical interface between the logic die and the memory dies within each stack. The interface from the host to each memory stack is an abstract, split-transaction, load/store-oriented one.

4. APPLICATIONS

We used 70 kernels from the sources described below for training and validating our ML-based performance prediction model. However, our energy model relies on average power measurements from GPU hardware which is currently only feasible in our framework for kernels with a run time of 1ms or longer. Therefore, we focus our application characterizations (and results discussion in Section 7) on kernels with a run time of 1ms or longer on our AMD Radeon HD 7970 native execution platform.

To provide an overall characterization of the application kernels, we use the following two metrics: sustained bandwidth usage and the number of vector ALU instructions executed per second. Figure 3 shows the kernels discussed in Section 7 in the two-dimensional space of the aforementioned metrics. For applications with multiple kernels, the kernel name is prepended with the application identifiers specified later in this section.

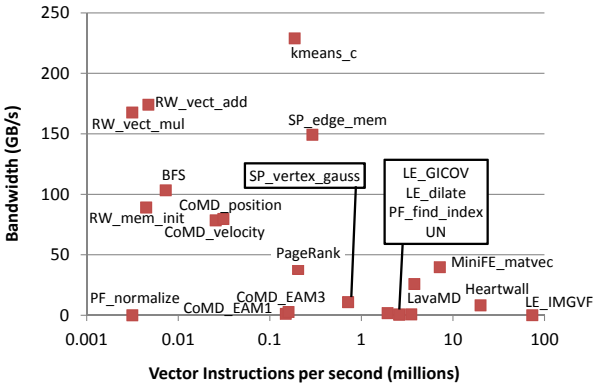

Figure 3: Characteristics of the application kernels

4.1 Graph Applications

We implemented a number of graph applications for this study. *Random walk* (RW) is a GPU implementation based on prior work by Gartner *et al.* [22]. Given a pair of graphs, it first performs random walks on both, then determines and counts the number of matching walks. *Shortest path* (SP) is a GPU implementation based on the prior work by Borgwardt *et al.* [13]. We first convert the input graphs into all pairs of shortest path graphs and compare them by applying a Gaussian kernel on the vertices and edges. *Unordered neighboring* (UN) determines similarities between nodes of two graphs by computing pair-wise similarities between nodes in their neighbor sets. *PageRank* is an algorithm proposed to prioritize web search results. Our implementation is based on recent work by Che *et al.* [17].

4.2 HPC Applications

The Mantevo Project [24] provides a collection of applications designed to mimic the characteristics of widely used high-performance computing algorithms. We study two of these. *MiniFE* represents the characteristics of larger applications modeling fluid and structural dynamics. Sparse matrix-vector multiplications dominate the execution time of the core computation of this benchmark and was chosen due to the importance of sparse matrix operations in HPC workloads. Codesigned Molecular Dynamics (*CoMD*) is a simplified molecular dynamics workflow. *CoMD* implements both Lennard-Jones and Embedded Atom Method (EAM) potential calculations. We use the EAM approach for its relevance for real HPC workloads.

4.3 Adapted Rodinia Benchmarks

Rodinia is a widely used GPGPU benchmark suite intended to be a representative sample covering a wide range of workload characteristics [18]. We use it here to augment the application space covered by the other benchmarks described above. While all Rodinia benchmarks are used in training our performance model, only the benchmarks with at least one kernel with execution time longer than 1 ms that can be obtained by increasing the input data size without algorithmic modifications are presented in our results discussions. The benchmarks that fit this criterion are *Breadth-first search* (BFS), *Heartwall*, *Kmeans*, *Leukocyte* (LE), *particle filter* (PF), and *LavaMD*.

5. PIM PERFORMANCE MODEL AND VALIDATION

Two broad categories of performance models – structural and analytical – are commonly studied in the literature [38]. 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 long-running workloads. This is especially true for workloads with large data sets and irregular memory access patterns, which are of particular interest for PIM evaluations. Therefore, in this study, we develop an analytical performance modeling framework to study how throughput-oriented PIM systems would perform.

5.1 Performance Model

At a high-level, our simulation framework relies on analyzing an application's behavior on existing GPU hardware to predict the performance of that application on future GPU configurations, including in-memory implementations. The application under test is executed on a real GPU operating at a known configuration (*e.g.*, number of CUs, processor frequency, memory bandwidth). For each GPU kernel invoked, hardware performance counter statistics are gathered, characterizing the application as it runs. This yields both the current performance of the application as well as statistics that indicate how the kernels ran. The statistics gathered from the native execution for each kernel are then fed to an ML-based performance scaling model that predicts the per-

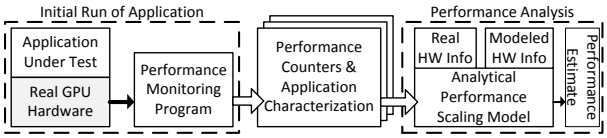

Figure 4: PIM performance modeling framework

formance at other machine configuration points of interest. The overall approach of this methodology is illustrated in Figure 4.

In essence, training kernels are analyzed under different machine configurations to ascertain their sensitivity to metrics such as parallelism, frequency, and memory bandwidth. Hardware performance counters are used to "fingerprint" these training runs. Later, the performance counters from a kernel under analysis will be used to find which training kernel it is most like. This, in turn, enables the model to predict the execution time on a GPU with a different mix of resources. For example, the real GPU used for native execution may be a 1GHz core with 32 CUs and 264GB/s of bandwidth to DRAM. Using the runtime and hardware performance information of an application on this device, our model could, for example, estimate the performance of the same application on a GPU with 8 CUs running at 700MHz with 400GB/s of memory bandwidth.

We use an AMD Radeon HD 7970 GPU for native execution and AMD's CodeXL profiler to collect performance counter metrics from hardware. The specific statistics gathered using performance counters include the number of vector and scalar instructions executed, the number of memory access instructions, the number of local data share accesses, utilization factors of various execution units, cache hit rates and the amount of data accessed from DRAM.

Scaling Assumptions

The key component of the above performance prediction methodology is the model that consumes the data from the native execution and scales the execution time to other hardware configurations. With T_{native} as the measured execution time of an arbitrary kernel on a known, native hardware configuration HW_{native}, the goal of the model is to implement the scaling function m such that:

$$T_{predict} = m(T_{native}, HW_{native}, HW_{predict}, P_{native})$$

where P_{native} is the set of performance counters gathered during native execution, $HW_{predict}$ is the future hardware configuration for which performance is to be predicted, and $T_{predict}$ is the predicted execution time of the same kernel invocation on that future hardware.

In our framework, this model is built on the following assumptions: (1) the detailed microarchitecture of the CUs of $HW_{predict}$ is similar to that of HW_{native} and, therefore, kernel performance is correlated to coarse configuration parameters such as the number of CUs, processor frequency and memory bandwidth; (2) kernels with similar statistics across a broad range of performance counter readings have similar sensitivity to hardware parameters; and (3) the performance impact of varying multiple hardware parameters is separable and the cumulative impact can be approximated by scaling first for the variation of one parameter followed by subsequent scaling for additional parameters.

The rest of this section describes how the ML-based model is constructed off-line using a large amount of training data and how it is used online for performance prediction.

Off-line Learning

Our ML-based performance model is trained using 70 kernels over a discretized 3D grid of 162 operating points that are supported by the native hardware used for our experiments. The dimensions of the grid are the number of CUs (C), CU frequency (f), and memory bandwidth (B). The number of CUs is varied among 32, 16 and 8. The CU frequency is varied from 500 to 1000 MHz in steps of 100 MHz. Memory bandwidth is varied by changing the memory frequency from 500 to 1300 MHz in steps of 100 MHz. Each kernel from the training set is executed at each point on this hardware configuration grid and the kernel execution time and performance counter statistics are obtained. We derive a kernel *feature vector* based on the collected hardware counter statistics at each point in the grid. Once feature vectors are available for all grid points for a given kernel, they are normalized to account for the varying scales used in gathering different metrics. Subsequently, we group the feature vectors of one kernel as a feature array for this kernel.

At each point on the HW configuration grid we also compute a time ratio triplet:

$$(RT_C, RT_f, RT_B) = (\frac{T_{C1}}{T_{C0}}, \frac{T_{f1}}{T_{f0}}, \frac{T_{B1}}{T_{B0}})$$

where each component is a time ratio between neighboring points on the 3D HW grid with other two parameters held constant. For example, RT_C is a time ratio between two neighboring points with different numbers of CUs but with fixed CU frequency and memory bandwidth. Similarly RT_f is a time ratio between two neighboring CU frequency points with fixed number of CUs and memory frequency. And RT_B is a time ratio between two neighboring memory bandwidth points with fixed number of CUs and CU frequency. This results in a 3D time ratio matrix with each element being a vector consisting of these three time ratio components.

Next, we proceed to the key part of the off-line learning process:

- Time ratio matrices are clustered into a predefined number of n clusters (in our implementation, $n = 4$) using the *k-means* algorithm.
- The clustering process produces n centroids, each of which is a time ratio matrix and a corresponding feature array partition. This maps each kernel feature array to a "representative" time ratio matrix.
- Using leave-one-out evaluation to guide the refinement of the clustering, an inner loop is performed:
  - For each test feature vector we use the *k-nearest neighbor* classification with $k = 5$ to find the cluster to which that feature vector belongs.
  - The representative time ratio matrix of the chosen cluster centroid is selected. This matrix is used when performing the prediction because the application under test appears similar to the known applications.
  - The prediction calculation is performed subject to the assumptions described earlier in this section to provide a predicted processing time at each point on the HW configuration grid.

  - A relative error is calculated at each point of the grid by comparing predicted time with real measured execution time at that particular hardware configuration.
- The mean of the relative errors is calculated over all feature vectors for the set of time ratio clusters.

To improve the quality of the clustering, this off-line learning process is conducted a sufficiently large number of times to repeat the clustering process with a new set of randomly selected seeds. Each iteration returns one mean relative error. The iteration yielding the minimum error is selected as our learned clusters.

In summary, during the training phase, the above described model "learns" a classification of kernels into different types based on their scaling characteristics and the sensitivity of each type of kernel to individual hardware parameters. As described in the following subsection, this knowledge is then used to predict execution times of new kernels at future design points, resulting in a much more sophisticated model than linearly scaling performance with hardware parameters. Not only is this model sensitive to the fact that some kernels are not affected by some parameters (e.g., memory-bound kernels are insensitive to processor frequency), but it is also able to infer complex interactions between the scaling coefficients of different parameters.

Online Prediction

Once the ML-based performance prediction model is built, performance prediction of new kernels is performed in near-real-time. For an arbitrary application with arbitrary problem sizes running on a known baseline hardware configuration, we collect the performance counter statistics for each kernel invocation and convert the data to a feature vector to be normalized in the same way as the training set. The prediction algorithm is very similar to the body of the off-line inner loop. The normalized feature vector is classified to find a cluster that the new kernel is closest to. Based on that, the representative time ratio matrix is selected and a prediction calculation chain is performed over the selected time ratio matrix based on the baseline (native) HW configuration, the predicted HW configuration and the native execution time. The result is a predicted processing time for the same kernel with the same problem size running on the predicted HW configuration. It is important to note that this model allows performance prediction not just at different hardware configurations, but also for previously unseen kernels.

5.2 Performance Model Validation

The leave-one-out process described for training the model also provides a method for characterizing the prediction accuracy of the model. A kernel that was not used for training the model is executed at a point p in the grid of hardware points described in Section 5.1 and the performance model is used to predict the execution time of that kernel at the other 161 points of the grid. This process is repeated for all possible values of p on the grid for that kernel, resulting in 26,082 ($= 161 * 162$) predictions per kernel. Each of the predicted times are then compared to the measured execution time at the corresponding hardware configuration to calculate the prediction errors. While this does not tell us directly how accurate our estimation is when modeling hardware settings outside our current GPU's range, estimation mechanisms that have high accuracy in these tests are more likely to be accurate for other design points as well.

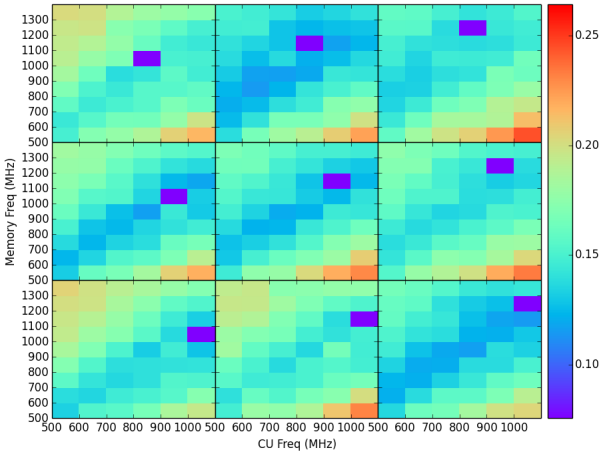

Figure 5: Prediction error (fraction) averaged over all benchmarks for nine native configurations. Purple rectangles are the native configurations and the color scale indicates the different estimation errors when predicting from that native point to other points.

Figure 5 shows a subset of the prediction errors of our model determined using the above approach, averaged over all of our benchmarks. Each of the nine tiles in the figure shows the prediction error using a different native execution point p indicated by the dark block. Figure 6 shows the average prediction errors for the kernels discussed in detail in this paper. The dotted line shows the average error (16.1%) across the full set of 70 kernels used in this work.

While our model is intended for near-real-time execution time prediction (after the one-time training overhead), the measured error of our model relative to hardware is competitive with errors observed for detailed simulators that run orders of magnitude slower. For example, Gutierrez *et al.* find 13% to 17% errors in correlating a custom-configured gem5 simulation framework to corresponding real hardware [23], while disabling the hardware features that gem5 couldn't model accurately. Weaver and McKee find errors of 24.6% and 67.6% for integer and floating point SPEC CPU benchmarks between SESC and real hardware [45]. Furthermore, even Yourst's evaluation of a highly-tuned, cycle-accurate simulator, PTLsim, shows 4.3% execution time prediction errors [48].

6. PIM ENERGY MODEL

Much like our performance model, our energy estimates are based on measuring power consumption on existing hardware on a per-kernel basis and projecting to future designs based on technology and system parameters. This power model is similar to the performance model shown in Figure 4. However, in addition to gathering performance counters, it also directly gathers power usage from the hardware.

The first step is to gather dynamic power data from our native execution platform – an AMD Radeon HD 7970 running at a core clock frequency of 1 GHz. AMD GPUs

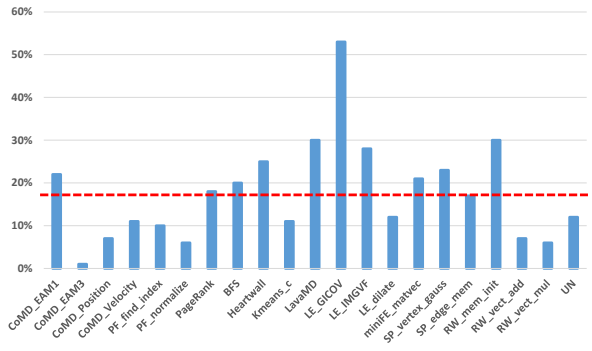

Figure 6: Individual benchmark prediction errors

Table 2. Dynamic power and memory energy estimates

	Baseline	Host		PIM	
Technology (nm)	28	22	16	22	16
Freq (MHz)	1000	1000	1000	650	650
Capacitance scaling	1.00	0.75	0.56	0.75	0.56
Vdd (V)	1.2	1.09	1.03	0.87	0.83
Dynamic CU power scaling	1.00	0.61	0.41	0.25	0.17
Memory energy (pJ/64b)	N/A	522	520	159	155

make chip-wide dynamic and static power available through memory-mapped registers that can be read by the host CPU. A host-side application polls these registers as GPU kernels are executed.

Because dynamic power usage of a chip is influenced by the operations executed on it (rather than just its temperature and voltage), it is important to have multiple dynamic power readings per computational kernel. Since we are able to scale performance only at the granularity of a kernel, we study the average power used over each kernel execution.

The dynamic power values read from the hardware encompass the GPU's die power, but do not include memories on the graphics card, physical interfaces, or other power drains at the periphery of the chip. Accounting for memory energy in our model is described later in this section.

The power values that we gather from existing hardware must be scaled to estimate the power that similar workloads would require on future systems. The two key components of this are scaling to future technologies (and corresponding operating points) and scaling to account for variations in the numbers of CUs.

Scaling to future technologies in our model is based on industry technology scaling projections and the ITRS [14, 7]. Based on those projections, we compute scaling factors for dynamic energy from current 28nm hardware to future design points of interest. Where our desired operating frequency f, as specified in Table 1, differs from those indicated by scaling projections, we adjust V_{dd} based on the relation $f \propto (V_{dd} - V_t)^2/V_{dd}$. Table 2 shows the relevant parameters and dynamic power scaling factors. We compute memory energy for future design points directly from access counts and, therefore, per-access energy from the baseline is not used in our projections.

Table 3. TDP variation normalized for voltage, frequency (MHz), and CU count

Design	CUs	Freq	Normalized TDP/CU
Radeon HD 7770	10	800	1.000
Radeon HD 7870	20	900	1.002
Radeon HD 7970	32	900	0.912
Radeon HD 7970	32	950	0.979

Our model scales design dynamic power (excluding memory and IO) linearly with the number of CUs. This assumes that other on-die resources outside the CUs scale with the number of CUs as well. We show TDP across four AMD GPU implementations from the same generation normalized for operating voltage, frequency and the number of CUs in Table 3. This data shows that TDP scales with CU count to within 10% across these designs and that linear scaling with CU count, therefore, is a reasonable approximation for an analytical model.

We base our static power estimates on the aggressiveness of the target design points for each processor's implementation. For host processors at 1GHz, we assume 30% of TDP as static power. For PIM targeting significantly less aggressive 650MHz operation, we assume 10% of TDP as leakage. We base these off scaled TDP to account for technology, operating point and configuration differences. We use this method instead of scaling the observed static power usage of application runs because static power readings are influenced by temperature, which is difficult to control between runs and may be different for future systems.

Finally, after estimating the power needed to run a computational kernel on a future GPU or PIM, the performance and power estimates are combined to yield energy estimates.

For host memory energy estimation, we assume SERDES power of 5pJ/b, at the low end of the HMC short reach PHY estimates [6]. We assume 7pJ per 64 bits of data for TSV traversal within the stack [4]. For both host and PIM, we assume 2pJ/b for DRAM access itself [27, 29]. We also incorporate factors accounting for wire traversal on the logic die (host or PIM) proportional to the square roots of the corresponding die areas. Table 2 shows the resulting per-access energy for a 64b word. These per access energy estimates are multiplied by dynamically observed access counts (from hardware performance counters) to compute total memory energy estimates. Even though off-chip memory interfaces often have high idle power overheads, we do not account for idle power consumption of host memory interfaces, which introduces a slight bias against the PIM configurations in our energy estimates.

As the objective of our study is to evaluate the energy efficiency of host computation relative to PIM computation, we assume the host is power-gated during PIM execution. Similarly, we assume the PIM logic is power-gated during host execution (aside from the memory access path) and does not contribute to system energy. We defer evaluating concurrent use of host and PIM to future work.

7. EVALUATION

We quantitatively evaluate the PIM architecture design choices described in Section 3 with the set of applications described in Section 4, using our performance and energy models.

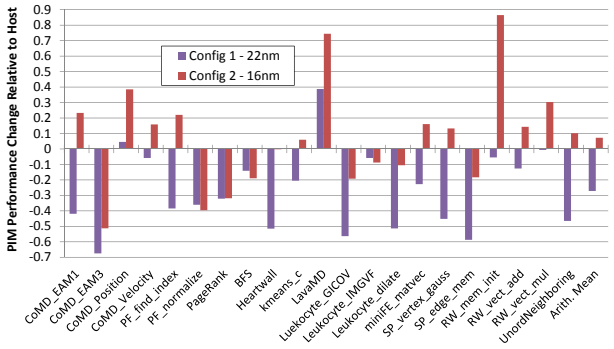

Figure 7: Performance change when executing on PIM, normalized to host performance. Positive values mean higher performance on PIM, while zero implies no performance change.

As discussed in Section 3.2, the characteristics of the design points evaluated here were determined based on past GPU architecture scaling trends and publicly available projections of future implementation technologies [7, 15]. We factored in both prevailing design and technology trends as well as hardware constraints such as area, power, and thermals. We believe this results in a distilled set of realistic design points for both the host and PIM processor implementations.

7.1 Performance Impact

Figure 7 shows the performance of each kernel on PIM normalized to the performance on the corresponding host. This reflects the performance tradeoff of running each kernel on PIM or host at a given technology node based on the configurations described in Table 1. We model each kernel executing on all in-memory processors (*i.e.*, two at 22nm and four at 16nm) in parallel except where noted otherwise later in this section.

As expected, in general, kernels with high compute requirements and low memory demands as characterized by Figure 3 suffer slowdowns on PIM at 22nm. The bandwidth advantages of PIM combined with improved relative PIM compute performance at 16nm (as a result of more CUs within the same 10W-per-stack budget) results in some of these, such as *CoMD_EAM1*, *SP_vertex_gauss* and *UN*, showing performance improvements over host execution. While the limited compute capability of PIM at 22nm results in slowdowns for most kernels, bandwidth-heavy kernels such as *CoMD_position*, *CoMD_velocity*, and *RW_mem_init* show significant performance gains on PIM at 16nm.

Balanced applications with high compute and bandwidth demands, such as *kmeans_c* and *SP_edge_mem* typically perform slower on PIM at 22nm, but gain performance with improved PIM compute capabilities at 16nm.

A number of benchmarks behave counter-intuitively to the characterization in Figure 3. *PF_normalize* and *PageRank* are dominated by synchronization and do not scale to multiple in-memory processors. Therefore, they are constrained to running on a single PIM instance, which results in degraded performance relative to host execution. *LE_IMGVF* does not have sufficient parallelism at the application level to utilize PIM across multiple memory stacks, and is also constrained to run on a single in-memory processor. *BFS*, while bandwidth-heavy, is bottlenecked by L1 cache bandwidth due to uncoalesced memory accesses and does not benefit from the improved bandwidth afforded by PIM. Conversely, *LavaMD*, while not bandwidth-intensive, is limited by L2 cache bandwidth and memory latency. As L2 cache bandwidth of GPU architectures are typically correlated to memory system performance (and not processor performance), our model predicts performance benefits for LavaMD from the improved memory system of PIM. *PF_find_index* and *Heartwall* are also L2 bandwidth bound to a lesser extent and is predicted to benefit from high memory system performance of PIM, especially at 16nm. *RW_vect_add* and *RW_vect_mul*, while bandwidth intensive, become constrained by the greatly reduced execution resources available on the 22nm PIM configuration but show speedups over host at the 16nm configuration.

These observations show that while a characterization based on application compute and bandwidth requirements can provide relevant intuitions about computations that may benefit from PIM, deeper analysis and modeling is necessary to fully understand the application characteristics that actually do benefit from the multi-dimensional architectural heterogeneity that exists between PIM and host implementations of even the same underlying microarchitecture.

7.2 Energy Impact

While in-memory processing can help the performance of some applications (especially those that can utilize the added bandwidth), compute-bound applications may see little performance benefit, or even slowdowns, when running their GPGPU computations in memory. Nonetheless, these applications may see energy efficiency benefits.

PIM in our system offers two avenues for these efficiency gains. First, accessing the memory system uses less energy, which directly benefits the energy efficiency of memory-heavy applications. Just as important, however, is the less aggressive processor design used for PIM – they can operate at a much lower frequency and use a less leaky design process while still offering good performance due to their added bandwidth. We found that the in-memory processors in our system always used less energy during our tests.

To quantify the energy impact of our PIM design, we present energy-delay product (EDP) and energy-delay^2 (ED^2) differences between running our benchmarks in memory versus running them on the host. Figure 8 presents the EDP for running the benchmarks on the in-memory processors in 22nm and 16nm designs normalized to the EDP of the host. Figure 9 details the ED^2. In almost all 22nm tests, the PIM's reduced energy yields a better EDP, even for applications that run slower (such as *PF_find_index* and *SP_edge_mem*).

ED^2 more heavily weights performance instead of energy, and Figure 9 shows that, at 22nm, many of the applications that lost performance when moved to PIM are also better run on the host when optimizing ED^2. Nonetheless, at 22nm, PIM is often a more energy-efficient option compared to running on the host.

This trend is further demonstrated at the 16nm design point. In this case, PIM has the benefit, as demonstrated in Section 7.1, of being much more performance competitive with the host. The 16nm node shows that the EDP and ED^2 of PIM is always better than the host, except for the ED^2 of *CoMD_EAM3*. This benchmark is extremely compute intensive, and gains little from in-memory execution.

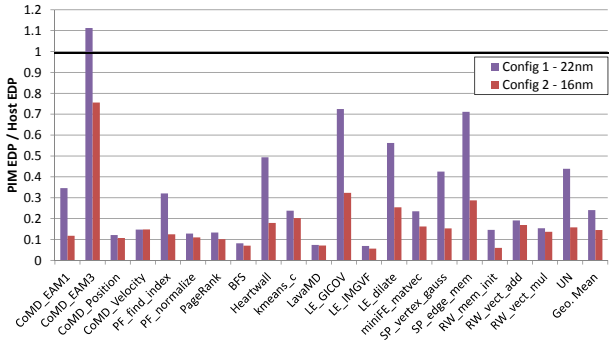

Figure 8: Relative energy delay product

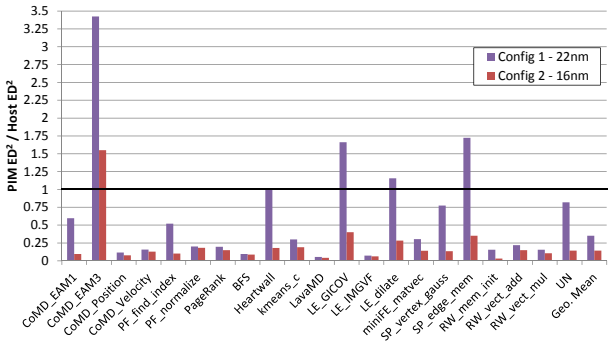

Figure 9: Relative energy-delay^2 (ED^2)

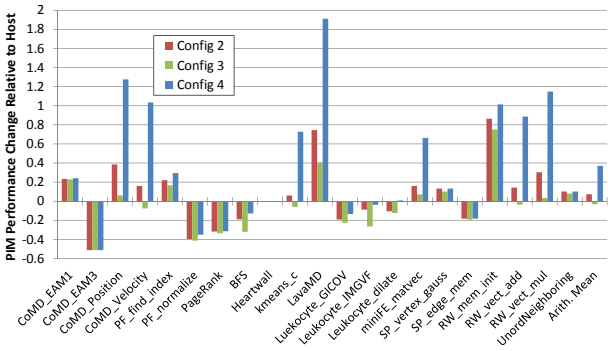

Figure 10: Performance change when executing on PIM, normalized to host performance at 16nm

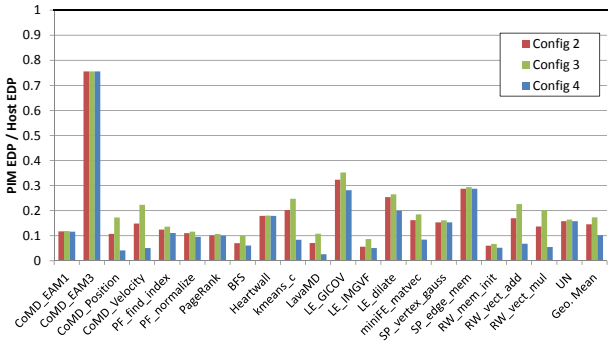

Figure 11: 16nm relative energy delay product

It's worth noting that these energy analyses included core and memory system power, but did not include system power (such as hard disks). These would decrease the energy efficiency benefit of PIM for programs that show a performance loss. However, the difference in power between using the host and PIM was often between 50-70W. As such, the difference when including the rest of the system (which may be blades or diskless HPC servers) will still likely benefit PIM in many cases. Conversely, for applications that run faster on PIM, the energy efficiency benefits when considering the full system may be even greater than reflected in our analyses.

In order to study the sensitivity of the above results to the relative bandwidth ratio between host and PIM, we analyze the two additional configurations described in Table 4. Config 3 is identical to Config 2 but with half the bandwidth for PIM and Config 4 is identical to Config 2 but with half the bandwidth for the host.

Table 4. Configurations to study bandwidth sensitivity

	Config. 3	Config. 4
Technology node (nm)	16	16
Host		
DRAM BW (GB/s)	640	320
DRAM BW/stack (GB/s)	160	80
Per-stack PIM		
DRAM BW per stack (GB/s)	320	640

7.3 Sensitivity to Bandwidth

Figure 10 shows the relative performance change of PIM for Config 2, 3, and 4 normalized to the corresponding hosts. Kernels that are heavily compute-limited (*e.g.*, *CoMD_EAM3*) are insensitive to the bandwidth variations. However, the reduced bandwidth of Config 3 leads to lower PIM performance for many of the other kernels. Similarly, decreasing the bandwidth of the host in Config 4 results in very significant slowdowns on the host for bandwidth-sensitive kernels.

Figures 11 and 12 show the normalized EDP and ED^2 for Configs 2, 3, and 4. These results show that, even if the in-memory processors were constrained by more pessimistic bandwidth considerations (as in Config 3), the energy benefits of moving the computation into the memory system still hold. Energy efficiency is reduced for these applications because the PIM performance benefits are reduced, but the 3D stacking and less aggressive designs still yield EDP and ED^2 benefits. The more constrained host of Config 4, which reduces the host's performance, again demonstrates this point.

8. FUTURE DIRECTIONS

This work represents a characterization of the tradeoffs of a specific type of processor (the GPU component of an APU) which we believe to be a good fit for in-memory computation. A thorough evaluation of the design space for in-memory processors using 3D die stacking requires a broad exploration encompassing other forms of programmable processors as well as configurable and fixed function processors. Further, this study evaluates the performance and energy efficiency of executing on PIM or host. Other usage models, such as offloading some computations to PIM thereby freeing up the host for other forms of computation, may provide additional benefits and need to be explored.

We have also assumed a generic GPGPU microarchitecture and memory hierarchy for our in-memory processors in this initial study. Optimization of the architecture and

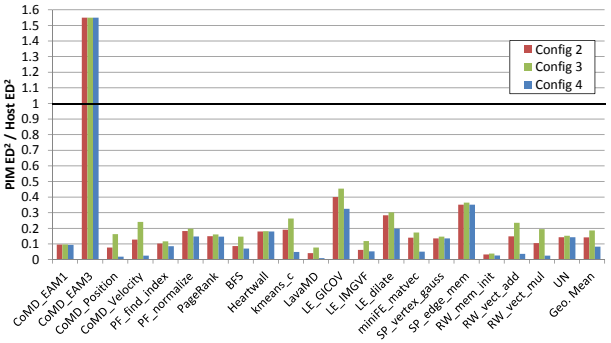

Figure 12: 16nm relative energy-delay^2

microarchitecture specifically for in-memory processing as well as specialization of communication and synchronization mechanisms present other rich areas for future research. A further area is the impact on host processor architecture if memory-intensive code segments are offloaded to in-memory processors.

As we adopt GPUs and APUs as the PIM processors, augmenting existing programming models (such as OpenCL) with an API to control data placement and compute dispatch suffices for the hardware design space evaluation presented in this paper. However, programmability for future systems with PIM components is a key area that requires further significant research. In particular, more research is needed on high-level abstractions to express application-level locality and data-compute affinity that enable efficient mapping of data and compute to memory stacks without significant programmer effort.

9. RELATED WORK

PIM attracted significant attention in the research community for a short period around the beginning of this century. Many of those efforts focused on one of two approaches. Efforts such as IRAM [21] integrated embedded DRAM on logic chips. However, this approach could not cost-effectively accommodate sufficient memory capacity for mainstream or high-performance systems due to the reduced density of embedded DRAM and the increased cost-per-bit of memory implemented in cutting-edge processor technologies. Efforts such as ActivePages [34], DIVA [20] and FlexRAM [26] 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 very low performance or highly specialized architectures geared only for select operations. This in turn limited the applicability and programmability of such PIM designs which, along with the cost implications of reduced DRAM density due to the presence of compute logic, limited the adoption of such approaches.

Integrating processors and memory using 3D die stacking, as we evaluate in this paper, does not provide as tight a coupling as integration on a single die. However, this approach enables the in-memory processors to be implemented on separate dies using logic processes and do not require significant changes to commodity memory dies geared towards emerging stacked memory standards. Further, as the in-memory processors are implemented in logic processes, they can be variants of existing processor designs which enable them to easily support familiar programming models while providing sufficient performance to support a broad range of applications. A similar approach has been advocated in a recent FlexRAM retrospective by J. Torrellas [42]. Balaprakash *et al.* [9] presented a study on HPC application characteristics and an analytical evaluation of their ability to exploit the increased bandwidth available to PIM. While they focused on extrapolating application characteristics to Exascale levels, we emphasize the hardware aspects of PIM in order to determine what PIM design points would result in performance, power, and thermal benefits. We demonstrate that a throughput-oriented processor (GPU) in the memory can effectively utilize the added bandwidth, benefitting a wide range of applications both in and outside of the HPC domain. We demonstrate that, while exascale computing may benefit from PIM, PIM does not require exascale workloads to be useful.

Sampson *et al.* [41] proposes a 3D stacked processor organization for accelerating 3D ultrasound beamformation. Pugsley *et al.* propose specialized processors on the base logic layer of HMC-like memory stacks to accelerate MapReduce workloads [39]. Our work differs from these recent efforts in focusing on programmable processors that support well-understood programming models and are broadly applicable across a variety of application domains instead of catering to a specific application domain. Another recent effort, Micron's Automata Processor, incorporates the ability to perform parallel automata processing within the memory arrays on the DRAM dies themselves [19]. However, this is also geared to a specific class of algorithms and further requires a specialized language to program.

10. CONCLUSION

Reducing off-chip data movement is becoming increasingly important for performance and energy efficiency. PIM has the potential to provide significant reductions in off-chip traffic. In this paper, we have presented an architecture for programmable, GPU-accelerated, in-memory processing implemented using 3D die-stacking and an evaluation of the viability of such an architecture. The throughput-oriented nature of GPU architectures is able to efficiently utilize the high bandwidth made available by vertically stacking in-memory processors directly under memory dies while providing the programmability needed to support a broad range of applications. The design points we evaluated were carefully chosen to be within the power and thermal constraints of 3D memory stacks, further establishing the feasibility of GPU-based PIM in near-future technology nodes.

We evaluated the chosen PIM design points using analytical performance and energy models that extrapolate to future design points using data gathered from hardware performance counters during native execution on existing hardware. Our evaluations show that PIM can provide performance and/or energy benefits for a variety of applications spanning a wide spectrum of compute/bandwidth ratios. On average, across the benchmarks evaluated, viable PIM configurations are shown to provide significant improvements in energy efficiency over representative mainstream configurations (76% reductions in EDP) with moderate performance losses (27% slowdown) at the 22nm technology node. At 16nm, viable PIM configurations are shown to provide marginal performance gains (7%) while providing even greater

energy efficiency improvements (85% reductions in EDP) on average. While many research areas remain to be explored in the broader context of PIM, our results demonstrate considerable promise in improving overall energy efficiency and performance of memory-limited applications.

11. ACKNOWLEDGEMENTS

We would like to thank Yasuko Eckert and Wei Huang for their input on modeling memory stack thermals. We appreciate the invaluable comments from the anonymouse reviewers.

12. REFERENCES

[1] `www.jedec.org/standards-documents/docs/jesd229`.

[2] `www.jedec.org/standards-documents/docs/jesd235`.

[3] `www.micron.com/products/hybrid-memory-cube`.

[4] ITRS interconnect working group, 2012 update. `www.itrs.net/links/2012Summer/Interconnect.pptx`.

[5] Elpida begins sample shipments of ddr3 sdram (x32) based on tsv stacking technology. `www.elpida.com/en/news/2011/06-27.html`, 2011.

[6] Initial hybrid memory cube short-reach interconnect specification issued to consortium adopters. Denali Memory Report, August 2012.

[7] *International Technology Roadmap for Semiconductors, 2011 Edition*. 2012 update.

[8] AMD. White paper: AMD graphics cores next (GCN) architecture. Jun 2012.

[9] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris. Exascale workload characterization and architecture implications. In *Proceedings of the High Performance Computing Symposium*, 2013.

[10] R. Balasubramonian. Exploiting 3D-stacked memory devices. IBM Research seminar, October 2012.

[11] B. Black. Die stacking is happening! In *46th IEEE/ACM International Symposium on Microarchitecture Keynote*, 2013.

[12] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCauley, P. Morrow, D. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking 3D microarchitecture. In *International Symposium on Microarchitecture*, 2006.

[13] K. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *5th International Conference on Data Mining*, 2005.

[14] S. Borkar. The exascale challenge. Keynote Presentation, 2010 Asia Academic Forum, Nov 2010.

[15] S. Borkar. Exascale computing – a fact or a fiction? Keynote Speech, 27th International Parallel & Distributed Processing Symposium, 2013.

[16] D. Chang, G. Byun, H. Kim, M. Ahn, S. Ryu, N. Kim, and M. Schulte. Reevaluating the latency claims of 3D stacked memories. In *18th Asia and South Pacific Design Automation Conference*, 2013.

[17] S. Che, B. M. Beckmann, S. K. Reinhard, and K. Skadron. Pannotia: Understanding irregular GPGPU graph applications. In *International Symposium on Workload Characterization*, 2013.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *International Symposium on Workload Characterization*, 2009.

[19] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes. An efficient and scalable semiconductor architecture for parallel automata processing. *To appear in IEEE Transactions on Parallel and Distributed Systems*.

[20] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, and G. Daglikoca. The architecture of the DIVA processing-in-memory chip. In *16th International Conference on Supercomputing*, 2002.

[21] B. R. Gaeke, P. Husbands, X. S. Li, L. Oliker, K. A. Yelick, and R. Biswas. Memory-intensive benchmarks: IRAM vs. cache-based machines. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2002.

[22] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, volume 2777, pages 129–143. 2003.

[23] A. Gutierrez, J. Pusdesris, R. Dreslinski, T. Mudge, C. Sudanthi, C. Emmons, and N. Paver. Sources of error in full-system simulation. In *International Symposium on Performance Analysis of Systems and Software*, 2014.

[24] D. Heroux, M.A.and Doerfler, P. Crozier, J. Willenbring, H. Edwards, A. Williams, M. Rajan, E. Keiter, H. Thornquist, and R. Numrich. Improving performance via mini-applications. Technical report, SAND2009-5574, 2009.

[25] D. Jevdjic, S. Volos, and B. Falsafi. Die-stacked DRAM caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache. In *40th International Symposium on Computer Architecture*, 2013.

[26] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas. FlexRAM: toward an advanced intelligent memory system. In *International Conference on Computer Design*, 1999.

[27] S. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the future of parallel computing. *Micro, IEEE*, 31(5):7–17, 2011.

[28] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. PicoServer: using 3D stacking technology to enable a compact energy efficient chip multiprocessor. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[29] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable DRAM alternative. *SIGARCH Computer Architure News*, 37(3):2–13, 2009.

[30] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, 2003.

[31] G. Loh. 3D-stacked memory architectures for multi-core processors. In *35th International Symposium on Computer Architecture*, 2008.

[32] G. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood, and K. Banerjee. A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy. In *43rd Design Automation Conference*, 2006.

[33] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz. Towards energy-proportional datacenter memory with mobile DRAM. *SIGARCH Computer Architure News*, 40(3):37–48, 2012.

[34] M. Oskin, F. Chong, and T. Sherwood. Active pages: a computation model for intelligent memory. In *25th Annual International Symposium on Computer Architecture*, 1998.

[35] Y. Y. Pan and T. Zhang. Improving VLIW processor performance using three-dimensional (3d) DRAM stacking. In *20th International Conference on Application-specific Systems, Architectures and Processors*, 2009.

[36] D. Patterson. Why latency lags bandwidth, and what it means to computing. Keynote Address, Workshop on High Performance Embedded Computing, 2004.

[37] J. T. Pawlowski. Hybrid memory cube (HMC). In *Hot Chips 23*, 2011.

[38] S. Pllana, I. Brandic, and S. Benkner. Performance modeling and prediction of parallel and distributed computing systems: A survey of the state of the art. In *1st International Conference on Complex, Intelligent and Software Intensive Systems*, 2007.

[39] S. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li. Ndc: Analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads. In *International Symposium on Performance Analysis of Systems and Software*, 2014.

[40] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. W. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. 36th International Symposium on Computer Architecture, 2009.

[41] R. Sampson, M. Yang, S. Wei, C. Chakrabarti, and T. Wenisch. Sonic millip3De: A massively parallel 3D-stacked accelerator for 3D ultrasound. In *19th International Symposium on High Performance Computer Architecture*, 2013.

[42] J. Torrellas. FlexRAM: Toward an advanced intelligent memory system: A retrospective paper. In *30th International Conference on Computer Design*, 2012.

[43] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM design and organization for energy-constrained multi-cores. In *37th International Symposium on Computer Architecture*, 2010.

[44] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *43rd International Symposium on Microarchitecture*, 2010.

[45] V. M. Weaver and S. A. McKee. Are cycle accurate simulations a waste of time? In *The Annual Workshop on Duplicating, Deconstructing, and Debunking*, 2008.

[46] A. White. Exascale challenges: Applications, technologies, and co-design. In *From Petascale to Exascale: R&D Challenges for HPC Simulation Environments ASC Exascale Workshop*, March 2011.

[47] D. H. Woo, N. H. Seong, D. Lewis, and H.-H. Lee. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In *IEEE 16th International Symposium on High Performance Computer Architecture*, 2010.

[48] M. Yourst. PTLsim: A cycle accurate full system x86-64 microarchitectural simulator. In *Performance Analysis of Systems Software International Symposium on*, pages 23–34, 2007.