Design and Evaluation of a Processing-in-Memory Architecture for the Smart Memory Cube

Erfan Azarkhish[1(✉)], Davide Rossi[1], Igor Loi[1], and Luca Benini[1,2]

[1] University of Bologna, Bologna, Italy
{erfan.azarkhish,davide.rossi,igor.loi}@unibo.it
[2] Swiss Federal Institute of Technology in Zurich, Zurich, Switzerland
luca.benini@iis.ee.ethz.ch

Abstract. 3D integration of solid-state memories and logic, as demonstrated by the Hybrid Memory Cube (HMC), offers major opportunities for revisiting near-memory computation and gives new hope to mitigate the power and performance losses caused by the "memory wall". Several publications in the past few years demonstrate this renewed interest. In this paper we present the first exploration steps towards design of the Smart Memory Cube (SMC), a new Processor-in-Memory (PIM) architecture that enhances the capabilities of the logic-base (LoB) die in HMC. An accurate simulation environment called SMCSim has been developed, along with a full featured software stack. The key contribution of this work is full system analysis of near memory computation including high-level software to low-level firmware and hardware layers, considering offloading and dynamic overheads caused by the operating system (OS), cache coherence, and memory management. A zero-copy pointer passing mechanism has been devised to allow low overhead data sharing between the host and the PIM. Benchmarking results demonstrate up to 2X performance improvement in comparison with the host System-on-Chip (SoC), and around 1.5X against a similar host-side accelerator. Moreover, by scaling down the voltage and frequency of PIM's processor it is possible to reduce energy by around 70 % and 55 % in comparison with the host and the accelerator, respectively.

Keywords: Processor-in-memory · Hybrid memory cube · Full system analysis · Software stack · Device driver

1 Introduction

The recent technological breakthrough represented by HMC is on its way to improve bandwidth, power consumption, and density [13]. This is while heterogeneous 3D integration also provides another opportunity for revisiting near memory computation to fill the gap between the processors and memories even further. Several research efforts in the past years demonstrate the renewed interest in moving part of the computation to where the data resides [4,5,11,22],

© Springer International Publishing Switzerland 2016
F. Hannig et al. (Eds.): ARCS 2016, LNCS 9637, pp. 19–31, 2016.
DOI: 10.1007/978-3-319-30695-7_2

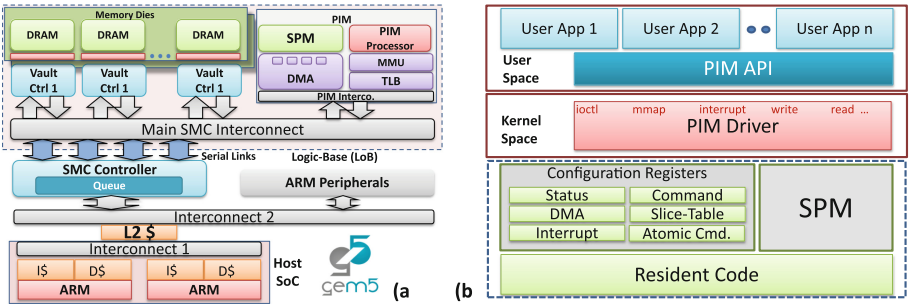

Fig. 1. An overview of the SMCSim environment for design space exploration of the smart memory cube (a), PIM's software stack (b).

specifically, in a 3D stacked memory context. Near memory computation can provide two main opportunities: (1) reduction in data movement by vicinity to the main storage resulting in reduced memory access latency and energy, (2) higher bandwidth provided by Through Silicon Vias (TSVs) in comparison with the interface to the host limited by the pins. Most recent works exploit the second opportunity by trying to accelerate data-intensive applications with large bandwidth demands [5]. In [22] and [4] also, networks of 3D stacked memories are formed and host processors are attached to their peripheries, providing even more hospitable platforms for processing-in-memory due to huge bandwidth internal to the memory-centric networks. These platforms, however, are highly costly and suitable for high-end products with extremely high performance goals [4]. Also, a look at the latest HMC Specifications [2] reveals that its ultra-fast serial interface is able to deliver as much bandwidth as is available in the 3D stack (Four serial links, 32 lanes each operating from 12.5 Gb/s to 30 Gb/s). For this reason, the same bandwidth available to a PIM on LoB is also theoretically available to the external host, and high-performance processing clusters or GPU architectures executing highly parallel and optimized applications can demand and exploit this huge bandwidth [25]. This puts PIM in a difficult but realistic position with its main obvious advantage over the external world being vicinity to the memory (lower access latency and energy) and not an increased memory bandwidth.

In this paper, we focus on this dark corner of the PIM research, and try to demonstrate that even if delivered bandwidth to the host can be as high as the internal bandwidth of the memory, PIM's vicinity to memory itself can provide interesting opportunities for energy and performance optimization. We focus on a scenario with a single PIM processor competing with a single thread on the host. In this scenario the caches of the host are not thrashed, the memory interface is not saturated, and the host can demand as much bandwidth as it requires. This is a worst-case scenario from PIM's point of view. Our PIM proposal (called the Smart Memory Cube) is built on top of the existing HMC standard with full compatibility with its IO interface specification. We have developed a full-system simulation environment called SMCSim and verified its accuracy against a Cycle-Accurate (CA)

model [8]. We devised an optimized memory virtualization scheme for zero-copy data sharing between host and PIM; enhanced PIM's operations by the aid from atomic processing in-memory operations; and enhanced PIM's memory access by means of a flexible Direct Memory Access (DMA) engine. Up to 2X performance improvement in comparison with the host SoC, and around 1.5X against a similar host-side accelerator are achieved. Power consumption was reduced by about 70 % and 55 % in comparison with the host and the accelerator, respectively, when PIM's voltage and frequency were scaled down. Lastly, the overheads associated with offloading code and data from host to PIM were found to be negligible. Related works are presented in Sect. 2. SMCSim is introduced in Sect. 3. Design of PIM and its software stack is described in Subsect. 3.1. Experimental results and conclusions are presented in Sects. 4 and 5.

2 Related Works

The design space for near memory computation is enormous, and several different concerns need to be addressed such as the micro-architecture of the PIM processor, memory management scheme, and communication mechanisms with the host [17]. In [11] a Coarse-grain Reconfiurable Accelerator (CGRA) is located on a separate die and connected to the DRAM dies through TSVs. Segmented memory without caching is used and 46 % energy saving along with 1.6X performance gain for Big Data applications are reported. The main difference between our work and this work is flexible support for virtual memory as well as considering the offloading overheads in our analysis. Active Memory Cube (AMC) [22] extends the logic layer of the HMC with clusters of vector processors without caches. Hardware coherence is maintained with the host and virtual memory support has been provided. 2X performance improvement is reported for dense matrix operations, increasing to 5X when vicinity aware memory allocation is utilized. Tesseract [4] features a network of memory cubes each accommodating 32 in-order cores with L1 caches and two prefetchers, optimized for parallelizing the PageRank algorithm. Uncacheable regions are shared with PIM and segmented memory without paging is supported. Up to 10X performance improvement and 87 % energy reduction has been provided in comparison with high-performance server hosts. Unlike these works, we focus on a context which external memory interface is not bandwidth saturated and PIM's benefits are determined only by latency. In addition, we utilize atomic commands and consider the offloading overheads in our studies, as well. In [5], the memory stack has been augmented with low level atomic in-memory operations. Host instructions are augmented, and full virtual memory support and hardware cache coherence is provided. for Big Data workloads up to 20 % performance and 1.6X energy gain is obtained. The main difference between our proposal and this work is that our PIM supports flexible execution of different computation kernels and its acceleration is not limited to the atomic operations only. Moreover, our solution is not dependent on the ISA of the host and with a proper software any host platform can communicate with it through its memory-mapped interface.

In this paper the goal is to provide flexible near-memory computation with full virtual memory support through a full-featured software stack compatible with the Parallel Programming Application Programming Interfaces (API). This is the first PIM effort to accurately model all layers from high level user applications, to low level drivers, OS, and hardware. Besides, a comprehensive accuracy verification versus a CA model has been performed which improves the quality of the obtained results.

3 The SMC Simulation Environment

SMCSim is a high-level simulation environment developed based on gem5 [12], capable of modeling an SMC device attached to a complete host SoC. Figure 1a illustrates an example of one such platform modeled in this environment. SMCSim has been designed based on gem5's General Memory System model [12] exploiting its flexibility, modularity, and high simulation speed, as well as, features such as check-pointing and dynamic CPU switching. An extensive comparison with a CA model [8] has been performed, as well, which confirms its reasonable accuracy. Figure 1a highlights the most important components in this environment. The host is a Cortex-A15 SoC capable of booting a full-featured OS. Inside the SMC, the vault controllers and the main interconnect are modeled using pre-existing components and tuned based on the CA models in [8]. A PIM is located on the LoB layer with flexible and generic computational capabilities. This configuration is completely consistent with the current release of the HMC Specification [13]. Also, SMC is not dependent or limited to any ISA and it is exposed to the host via memory mapped regions. Therefore, any host platform should be able to communicate with it by the aid from a proper software stack. For the serial links: bandwidth, serialization latency, and packetization overheads are obtained from [2]. They accept the same address ranges and each packet can travel over any of them [2,3]. SMC/HMC controller is responsible for translating the host protocol (AXI for example) to the serial links protocol. Plus, it should have large internal buffers to hide the access latency of the cube. Different sources of latency are modeled in this environment and calibrated in Sect. 4.

3.1 Design of PIM and Its Software Stack

We have chosen an ARM Cortex-A15 core without caches or prefetchers for PIM, and augmented it with low cost components to enhance its capabilities. Our choice of ARM is because it offers a mature software stack, its system bridges (AXI) are well understood, and it is an energy-efficient architecture. Nevertheless, the architecture is not limited to it. As shown in Fig. 1a, PIM is attached to the main LoB interconnect through its own local interconnect, and features a Scratchpad Memory (SPM), a DMA engine, a Translation Look-aside Buffer (TLB) along with a Memory Management Unit (MMU).

PIM has been designed to directly access user-space virtual memory. Its TLB serves for this purpose. Apart from memory protection benefits, it replaces

memory-copy from user's memory to PIM with a simple virtual pointer passing. Scalability and programmability are improved, and offloading overheads are reduced to a great extent. Since user's memory is paged in conventional architectures, PIM should support it, as well. To add more flexibility we introduce the concept of *slices* as a generalization to memory pages: a region of memory composed by 1 or more memory-pages which is contiguous in both virtual and physical memory spaces. With this definition, contiguous memory pages which map to contiguous page-frames can be merged to build larger slices, with arbitrary sizes. Note that, any change in page-size requires rebuilding the OS and all device drivers, while slices are transparent to the OS and other devices. PIM's memory management is done at the granularity of the slices. The first slice is devoted to PIM's SPM, and the rest of the memory space is mapped immediately after this region. Upon a TLB miss, a data structure in DRAM called the *slice-table* is consulted and the translation rules are updated on a Least Recently Used (LRU) basis. *Slice-table* is similar to page-tables and contains all translation rules for the computation kernel currently executing on PIM. It is built during the task offloading procedure by PIM's device-driver. Most host-side accelerators with virtual memory support rely on the host processor to refill the rules in their TLB. The OS consults its page-table to reprogram the IO-MMU of the device, and then wakes up the accelerator to continue its operation. Since PIM is far from the host processors, asking them for a refill upon every miss can result in a large delay. As an alternative, PIM's TLB contains a simple controller responsible for fetching the required rule from the slice-table. Apart from the performance benefits, this allows for fully independent execution of PIM.

A simple zero-load latency analysis (Sect. 4) reveals that the latency of directly accessing DRAM by PIM is harmful to its performance, therefore latency-hiding mechanisms are required. Caches and prefetchers provide a higher-level of abstraction without much control. This is desired for SoCs far from the memory and flexible enough to support different main memory configurations [6]. DMA engines plus SPMs, on the other hand, provide more control and can reduce energy consumption without much increase in the programming effort [18]. For this reason they make more sense in a near-memory processor. We have augmented PIM with a DMA engine capable of bulk data transfers between the DRAM vaults and its SPM (See Fig. 1a). It allows multiple outstanding transactions by having several DMA resources, and accepts virtual address ranges without any alignment or size restrictions. A complementary way to address this problem is to move some very specific arithmetic operations directly to the DRAM dies and ask the vault controller to do them "atomically". In-memory operations can reduce data movement when computation is local to one DRAM row [5], and they are easily implementable in the abstracted memory interface of HMC. We have augmented our vault controllers with three types of atomic commands suitable for the benchmarks under our study: *atomic-min*, *atomic-increment*, and *atomic-add-immediate*. These commands are implemented in the vault-controllers and their latency is hidden behind the DRAM timings. On the other side, instead of modifying PIM's ISA to support these commands, we added specific memory mapped registers to the PIM processor.

By configuring these registers PIM is able to send atomic operations towards the SMC vaults. For computations that need to gather information not fully localized to a single memory vault, DMA can be more beneficial. While, highly localized computations with low computational intensity are better performed as close as possible to the memory dies, by means of the atomic commands.

A software-stack has been developed for the user level applications to view PIM as a standard accelerator (See Fig. 1b). At the lowest level, a resident program runs on PIM performing the required tasks. A dynamic binary offloading mechanism has been designed to modify this code during runtime. PIM also features a set of configuration registers mapped in the physical address space and accessible by the host. PIM's device driver has been adopted from Mali GPU's driver [1] and is compatible with standard accelerators as well as parallel programming APIs such as OpenCL. This light-weight driver provides a low-overhead and high-performance communication mechanism between the API and PIM. An object-oriented user-level API has been designed, as well, to abstract away the details of the device driver and to facilitate user's interface. Offloading and coordinating the computations on PIM are initiated by this API.

PIM targets execution of medium sized computation kernels having less than a few kilobytes of instructions. The host processor parses the binary Executable and Linkable Format (ELF) file related to a precompiled computation kernel, and dynamically offloads *.text* and *.rodata* sections to PIM's memory map by the aid from the API. This procedure is called the *kernel-offloading*. On the other hand, the virtual pointer to preallocated user level data structures need to be sent to PIM for the actual execution to take place (*task-offloading*). PIM's API sends the page numbers associated with user data structures to the driver, and the driver builds the slice-table in the kernel memory space using the physical addresses. Next, caches are flushed and the virtual pointers are written to PIM's memory mapped registers. An interrupt is then sent to PIM to wake it up for execution. This mechanism prevents PIM from accessing unwanted physical memory locations, and allows it to traverse user-level data structures without any effort. A polling thread in PIM's API waits for completion of the offloaded task.

4 Experimental Results

Our baseline host system is composed of two Cortex-A15 CPU cores @2 GHz with 32 KB of instruction cache, 64 KB of data cache, and a shared 2 MB L2 cache with associativity of 8 as the last-level cache (LLC). The block size of all caches is increased to 256 B to match the row buffer size of the HMC model (effect of cache block size is studied later in this section). The memory cube model provides 512 MB of memory with 16 vaults, 4 stacked memory dies, and 2 banks per partition [8]. PIM has a single core processor similar to the host processors running at the same frequency, with the possibility of voltage and frequency scaling by means of dedicated clock and voltage domains on the LoB. Maximum burst size of PIM's DMA is set to 256 B by default. We performed a detailed accuracy comparison of the high-level SMC against a CA model [8].

We applied identical traffic patterns with various bandwidth demands to both CA and gem5-based models, and compared their delivered bandwidth and total execution time over a large design space defined by the architectural parameter. Several experiments demonstrated that total execution time and delivered bandwidth of the gem5-based model correlate well with the CA model: with low or medium traffic pressure, the difference was less than 1 %, and for high pressure saturating traffic the difference was bounded by 5 %, in all cases. Next, we calibrated the latency of the individual components based on the available data from the literature and the state of the art. The results are shown in Table 1.

Table 1. Zero-load latency breakdown of memory accesses from the host and PIM.

HOST: L2 cache refill latency - (Size: 256 B) [Total: 102.3 ns]			PIM: 4B read access latency (no caches) [Total: 39.1 ns]		
Membus	1 Cycle@2 GHz	Flit:64 b	PimBus	1 Cycle@1 GHz	Flit:32 b
SMCController	8 Cycles@2 GHz	Pipeline Latency [21]	SMCXbar	1 Cycle@ 1GHz	Flit:256 b [8]
SERDES	1.6 ns	SER = 1.6 DES = 1.6 [14]	Vault Ctrl. Front-end	4 Cycles@1.2 GHz	[8]
Packet Transfer	13.6 ns	16 Lanes@10 Gb/s, 128 b hdr [2]	tRCD	13.75 ns	Activate [14]
PCB Trace Latency	3.2 ns + 3.2 ns	Round Trip	tCL	13.75 ns	Issue Read Command [14]
SMCXBar	1 Cycle@1 GHz	Flit:256 b [8]	tBURST	3.2 ns	1 Beat [19]
VaultCtrl.frontend	4 Cycles@1.2 GHz	[8]	Vault Ctrl. Back-end	4 Cycles@1.2 GHz	[8]
tRCD	13.75 ns	Activate [14]	PimBus	1 Cycle@1 GHz	Flit:32 b
tCL	13.75 ns	Issue Read Command [8]			
tBURST	25.6 ns	256 B Burst [19]			
VaultCtrl.backend	4 Cycles@1.2 GHz	[8]			
SERDES	1.6 ns	SER = 1.6 DES = 1.6 [14]			
Packet Transfer	13.6 ns	16 Lanes@10 Gb/s, 128 b hdr [2]			
SMCController	1 Cycles@2 GHz	Pipeline Latency [21]			
Membus	1 Cycle@2 GHz	Flit:64 b			

Data intensive applications often categorized as “Big Data” workloads are widely chosen in the literature as the target for near memory acceleration [4,5,11]. The common trait shared by most of these applications is sensitivity to memory latency and high bandwidth demand. Graph traversal applications are an interesting example in this group due to their unpredictable memory access patterns and high ratio of memory access to computation [5]. A common use for these benchmarks is social network analysis. We have chosen four large-scale graph processing applications, and try to accelerate their main computing loop (i.e. the computation kernel) by offloading it to PIM for execution. Our results are also compared with simple matrix addition (MAT), as a reference with lowest

possible memory access latency sensitivity. Here is the list of the graph benchmarks studied in this paper: *Average Teenage Follower (ATF)* [5] counts for each vertex the number of its teenage followers by iterating over all teenager. We have implemented an *atomic-increment* command inside the memory cube for this purpose. *Breadth-First Search (BFS)* [5] visits the vertices closer to a given source vertex first by means of a FIFO queue. No atomic operations were utilized to accelerate this kernel, however, the queue for visiting the nodes has been implemented in PIM's SPM, given that in sparse graphs the required queue size of BFS is determined by the maximum outage degree of the nodes and not the number of nodes. *PageRank (PR)* [4] is a well-known algorithm that calculates the importance of vertices in a graph. We have implemented a single-precision floating point version of this kernel, plus an atomic floating-point *add-immediate* on the vault side. *Bellman-Ford Shortest Path (BF)* [5] finds the shortest path from a given source vertex to other vertices in a graph. Vault controllers have been augmented with *atomic-min* to facilitate its execution. We have used randomly generated sparse graphs ranging from 4 K node to 512 K nodes with characteristics obtained from real world data sets [15], and implemented them using the List of Lists (LIL) representation format [20], as its easily parallelizable and scalable to many nodes, and does not have the scalability limitations of the adjacency matrices. We use two DMA resources to efficiently hide the latency of traversing the LIL.

First we study offloading overheads. ATF has been executed on the host side and then offloaded to PIM. In Fig. 2a, *kernel-offload* represents the overhead associated with reading the ELF file from the secondary storage of the host, parsing it, and offloading the binary code (as explained in Subsect. 3.1); *task-offload* is all overheads associated with virtual pointer passing to PIM for the graph to be analyzed; and *host-execution* is the absolute execution time of the kernel on host. It can be seen that the task-offload overhead decreases with the size of the graph and is always below 5 % of the total execution time of the ATF. Most of this overhead is due to cache flushing and less than 15 % of it belongs to building the slice-table and pointer passing. Also, kernel-offload is usually performed once per several executions, therefore, its cost is amortized among them. Plus, for kernels like PR with several iterations relative overheads becomes even lower. Next, we analyze the speed-up achieved by PIM in terms of host's execution time divided by PIM's. The number of graph nodes has been changed and Fig. 2b (left vertical axis) illustrates the results. Lightly shaded columns represent PIM's execution without any aid from the atomic HMC commands, while the highly shaded ones use them. PIM's frequency is equal to the host (2 GHz). An average speed-up of 2X is observable across different graph sizes, and as the size increases, speed-up increases as well. This can be associated with increase in cache misses in the LLC of the host (plotted on the right vertical axis). Furthermore, the average benefit of using *atomic-increment*, *atomic-min*, *atomic-float-add* is obtained as 10 %, 18 %, and 35 %, respectively. A cache line size of 256 Bytes was mentioned before for both cache levels in the host. We can see in Fig. 2c that this choice has been in favour of the host in terms of performance,

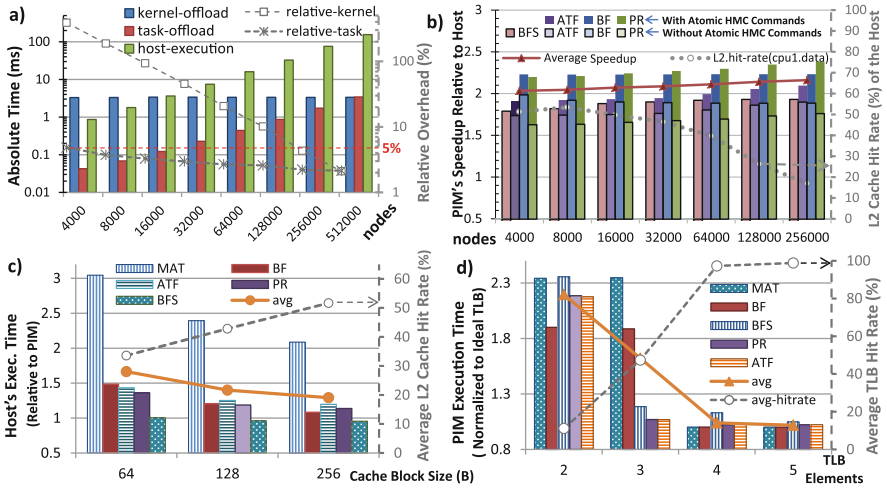

Fig. 2. Offloading overheads in execution of the ATF kernel (a), PIM's speed-up with/without SMC Atomic Commands [left axis] and LLC hit-rate associated with the data port of the executing CPU [right axis] (b), Host's execution time versus cache block size (c), and effect of PIM's TLB size on hit-rate and execution time (d) (Colour figure online).

leading to an increase in the LLC hit-rate. This is mainly due to sequential traversing of the graph nodes which results in a relatively high spatial locality (MAT in this plot represents simple 1000×1000 matrix addition as a reference for comparison). Two additional experiment on PIM identify its optimal TLB size (Fig. 2d) and the required DMA transfer size (Fig. 3a). Four TLB elements were found enough for the studied computation kernels and a nearly perfect TLB hit-rate was achieved. Also, transfer size was changed from 32 B to 512 B and the optimal point was found around 256 B transfers. This is because small transfers cannot hide memory access latency very well, and too large transfers incur larger queuing latency in the vault controllers, as the size of the row buffers is fixed at 256 B.

For estimation of the power consumption, we assumed 28 nm Low Power technology as the logic process. For the interconnects, energy/transaction was extracted from logic synthesis of the AXI-4.0 RTL model [8]. Cache power consumption was extracted from the latest release of CACTI [24]. Power consumed in the DRAM devices was extracted from DRAMPower [10]. The energy consumed in the vault controllers was estimated to be 0.75 pJ/bit [9]. SMC Controller was estimated to consume 10 pj/bit by scaling values obtained in [21]. For serial links, energy per transaction was considered 13.7 pj/bit [16], and an idle power consumption of 1.9 Watts (for transmission of the null flits) was estimated based on the maximum power reported in [13] and link efficiency in [19]. Also, since power state transition for the serial links introduces long sleep latency in the order of a few hundred nanoseconds, and a wakeup latency of a few microseconds [3], we assumed that during host's computations, links are

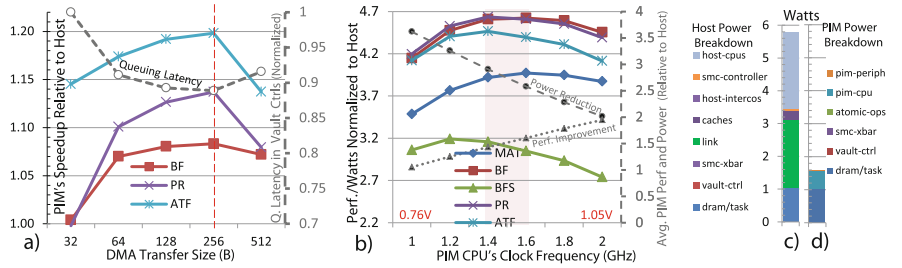

Fig. 3. Effect of DMA transfer size on PIM's speed-up (a), PIM's energy efficiency vs. its clock frequency (b), power breakdown for execution of the host (c) and PIM (d) (Colour figure online).

fully active, while when PIM starts computing, links can be power gated [3]. For the processors, percentage of active/idle cycles were extracted from gem5, and the power consumption for each one were estimated based on [23]. The energy consumed by the Floating Point operations was estimated based on [7], and the energy associated with atomic commands, PIM's TLB, and its DMA engine were estimated based on logic synthesis and plotted in Fig. 3d as "atomic-ops" and "pim-periphs".

To perform a fair analysis of the energy efficiency achieved by PIM, we omitted the system-background power and only considered the energy consumption related to the execution of tasks. Background power consists of all components which consume energy whether host is active or PIM, and can range from system's clocking circuits to peripheral devices, the secondary storage, the cooling mechanism, as well as, unused DRAM cells. One important observation was that for typical social graphs with less than 1 million nodes the total allocated DRAM size was always less than 100 MB (Using LIL representation). While, a significant amount of power is consumed in the unused DRAM cells. In fact, increasing stacked DRAM's size is one of the background sources which can completely neutralize the energy efficiency achieved by PIM. For this reason, we only consider the energy consumed in the "used" DRAM pages, all components in LoB of the memory cube, the serial links, the host processors, and their memory interface including the interconnects and the caches. Power consumption breakdown for execution of the task on PIM and the host are illustrated in Fig. 3c and d. The main contributors were found to be DRAM, the processors, and the serial links. The voltage and frequency of PIM's processor (on LoB) have been scaled down from 2 GHz@1.05 V to 1 GHz@0.76 V [23]. Under these circumstances, PIM can reduce power consumption by over 70 %. To find the optimal point in terms of energy efficiency we scaled the voltage and frequency of PIM's processor and plotted PIM's performance per watts (calculated as total execution time divided by consumed power) normalized to the host in Fig. 3b It can be seen that clocking PIM at around 1.5 GHz leads to highest energy efficiency in all cases.

As the final experiment, PIM was compared with a host-side accelerator with similar capabilities, to study the effect of vicinity to the main memory.

For this purpose, we detached the complete PIM subsystem from the cube and connected it to "Interconnect 2" (Fig. 1a) without any change in its capabilities. For matrix addition, no performance difference was observed, because the DMA engine effectively hides memory access latency. However, in all four graph traversal benchmarks PIM beats the host-side accelerator by a factor of 1.4X to 1.6X. This can be explained by the latency sensitivity of the graph traversal benchmarks (Average memory access latency from PIM to the main memory was measured as 46 ns, while for the host side accelerator this value had increased to 74 ns). Also, since the host side accelerator needs the serial links and the SMC Controller to be active, under the same conditions as the previous experiment (Considering power for the active banks of the DRAM and scaling down the voltage and frequency of PIM's processor), our PIM achieves an energy reduction of 55 % compared to a similar accelerator located on the host. Lastly, according to Little's law, the host side accelerator requires more buffering to maintain the same bandwidth, due to higher memory access latency. This results in increased manufacturing cost and energy.

5 Conclusions

In this paper we presented the first exploration steps towards design of SMC, a new PIM architecture that enhances the capabilities of the LoB die in HMC. An accurate simulation environment called SMCSim has been developed, along with a full featured software stack. A full system analysis considering offloading and all dynamic overheads demonstrated that even in a case where the only benefit of using PIM is latency reduction, up to 2X performance improvement in comparison with the host SoC, and around 1.5X against a similar host-side accelerator is achievable. Moreover, by scaling down the voltage and frequency of the proposed PIM it is possible to reduce energy by about 70 % and 55 % in comparison with the host and the accelerator, respectively.

Acknowledgment. This work was supported, in parts, by EU FP7 ERC Project MULTITHERMAN (GA no. 291125). We would also like to thank Samsung Electronics for their support and funding.

References

1. Mali-400/450 GPU device drivers. http://malideveloper.arm.com/resources/drivers
2. Hybrid memory cube specification 2.1 (2014). http://www.hybridmemorycube.org/
3. Ahn, J., Yoo, S., Choi, K.: Low-power hybrid memory cubes with link power management and two-level prefetching. IEEE Trans. Very Large Scale Integr. VLSI Syst. **99**, 1–1 (2015)
4. Ahn, J., Hong, S., Yoo, S., Mutlu, O., Choi, K.: A scalable processing-in-memory accelerator for parallel graph processing. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA 2015, pp. 105–117. ACM, New York, NY, USA (2015)

5. Ahn, J., Yoo, S., Mutlu, O., Choi, K.: PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA 2015, pp. 336–348. ACM, New York, NY, USA (2015)
6. Alves, M.A.Z., Freitas, H.C., Navaux, P.O.A.: Investigation of shared L2 cache on many-core processors. In: 2009 22nd International Conference on Architecture of Computing Systems (ARCS), pp. 1–10, March 2009
7. Aminot, A., Lhuiller, Y., Castagnetti, A., et al.: Floating point units efficiency in multi-core processors. In: Proceedings, ARCS 2015 - The 28th International Conference on Architecture of Computing Systems, pp. 1–8, March 2015
8. Azarkhish, E., Rossi, D., Loi, I., Benini, L.: High performance AXI-4.0 based interconnect for extensible smart memory cubes. In: Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition, DATE 2015, pp. 1317–1322. EDA Consortium, San Jose, CA, USA (2015)
9. Boroujerdian, B., Keller, B., Lee, Y.: LPDDR2 memory controllerdesign in a 28 nm process. http://www.eecs.berkeley.edu/bkeller/~rekall.pdf
10. Chandrasekar, K., Akesson, B., Goossens, K.: Improved power modeling of DDR SDRAMs. In: 2011 14th Euromicro Conference on Digital System Design (DSD), pp. 99–108, August 2011
11. Farmahini-Farahani, A., Ahn, J.H., Morrow, K., Kim, N.S.: NDA: near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 283–295, February 2015
12. Hansson, A., Agarwal, N., Kolli, A., et al.: Simulating DRAM controllers for future system architecture exploration. In: 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 201–210, March 2014
13. Jeddeloh, J., Keeth, B.: Hybrid memory cube new DRAM architecture increases density and performance. In: 2012 Symposium on VLSI Technology (VLSIT), pp. 87–88, June 2012
14. Kim, G., Kim, J., Ahn, J.H., Kim, J.: Memory-centric system interconnect design with hybrid memory cubes. In: 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 145–155, September 2013
15. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection, June 2014. http://snap.stanford.edu/data
16. Lloyd, S., Gokhale, M.: In-memory data rearrangement for irregular, data-intensive computing. Computer **48**(8), 18–25 (2015)
17. Nair, R.: Evolution of memory architecture. Proc. IEEE **103**(8), 1331–1345 (2015)
18. Paul, J., Stechele, W., Kroehnert, M., Asfour, T.: Improving efficiency of embedded multi-core platforms with scratchpad memories. In: 2014 27th International Conference on Architecture of Computing Systems (ARCS), pp. 1–8, February 2014
19. Rosenfeld, P.: Performance Exploration of the Hybrid Memory Cube. Ph.D. thesis, University of Maryland (2014)
20. Salihoglu, S., Widom, J.: GPS: A graph processing system. In: Proceedings of the 25th International Conference on Scientific and Statistical Database Management, SSDBM, pp. 22:1–22:12. ACM, New York, NY, USA (2013)
21. Schaffner, M., Gürkaynak, F.K., Smolic, A., Benini, L.: DRAM or no-DRAM? exploring linear solver architectures for image domain warping in 28 nm CMOS. In: Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition. DATE 2015, EDA Consortium (2015)

22. Sura, Z., Jacob, A., Chen, T., et al.: Data access optimization in a processing-in-memory system. In: Proceedings of the 12th ACM International Conference on Computing Frontiers. CF 2015, pp. 6:1–6:8. ACM, New York, NY, USA (2015)
23. Tudor, B.M., Teo, Y.M.: On understanding the energy consumption of ARM-based multicore servers. SIGMETRICS Perform. Eval. Rev. **41**(1), 267–278 (2013)
24. Wilton, S., Jouppi, N.: CACTI: an enhanced cache access and cycle time model. IEEE J. Solid-State Circuits **31**(5), 677–688 (1996)
25. Zhong, J., He, B.: Towards GPU-accelerated large-scale graph processing in the cloud. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), vol. 1, pp. 9–16, December 2013

http://www.springer.com/978-3-319-30694-0

Architecture of Computing Systems -- ARCS 2016
29th International Conference, Nuremberg, Germany,
April 4-7, 2016, Proceedings
Hannig, F.; Cardoso, J.M.P.; Pionteck, T.; Fey, D.;
Schröder-Preikschat, W.; Teich, J. (Eds.)
2016, XX, 402 p. 164 illus., Softcover
ISBN: 978-3-319-30694-0