Understanding Energy Aspects of Processing-near-Memory for HPC Workloads

Hyojong Kim[†] Hyesoon Kim[†] Sudhakar Yalamanchili[†] Arun F. Rodrigues[‡]

[†]Georgia Institute of Technology [‡]Sandia National Laboratories

{hyojong.kim,hyesoon.kim,sudha}@gatech.edu afrodri@sandia.gov

ABSTRACT

Interests in the concept of processing-near-memory (PNM) have been reignited with recent improvements of the 3D integration technology. In this work, we analyze the energy consumption characteristics of a system which comprises a conventional processor and a 3D memory stack with fully-programmable cores. We construct a high-level analytical energy model based on the underlying architecture and the technology with which each component is built. From the preliminary experiments with 11 HPC benchmarks from Mantevo benchmark suite, we observed that misses per kilo instructions (MPKI) of last-level cache (LLC) is one of the most important characteristics in determining the friendliness of the application to the PNM execution.

CCS Concepts

•Hardware → Power estimation and optimization; Emerging architectures;

Keywords

PNM; Energy Estimation; Offloading

1. INTRODUCTION

Power and energy consumption are the primary concerns for large-scale systems. The concept of processing-near-memory (PNM), which places computation into the location where data resides, has recently been getting a lot of attention because it is expected to reduce the energy consumption associated with data movement. We envision that one promising design for a PNM architecture is that of fully programmable cores, similar to GPU-like accelerators. In such a PNM system, offloading based programs such as OpenCL applications can be executed on PNM cores without any additional programmer's efforts. In this context, crucial questions are:

- What is the most important factor in deciding where the computation should be performed on?
- What kind of applications will get performance and/or energy benefits from PNM execution?

MEMSYS '15, October 05-08, 2015, Washington DC, DC, USA

© 2015 ACM. ISBN 978-1-4503-3604-8/15/10...$15.00

DOI: http://dx.doi.org/10.1145/2818950.2818985

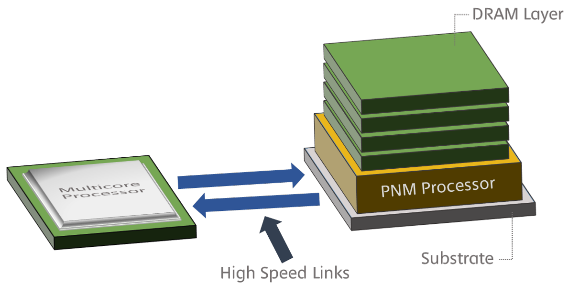

Figure 1: An envisioned PNM system where a host multicore processor is connected with a PNM stack. An application can run on the host processor as in the traditional manner, or it can be offloaded to the PNM processor where data can be accessed in a more efficient way.

To answer these questions, we construct a first-order energy model that provides insights on the trade-offs of PNM execution from energy aspects.

2. BACKGROUND

In this section, we provide an essential background on an Hybrid Memory Cube (HMC) based PNM architecture and the envisioned PNM system.

2.1 PNM architecture

Figure 1 shows our baseline PNM architecture. A PNM stack is composed of a number of autonomous *vaults*, each of which is roughly the equivalent of a channel in the traditional DDRx memory system. Each vault contains a number of autonomous *partitions*, each of which is located in different story of DRAM dies and consists of more than one independently operating memory *banks*. Assuming 16 vaults within a PNM stack and two banks per each partition, for instance, a total of 128 (in the case of four DRAM dies) or 256 (in the case of eight DRAM dies) concurrently operating banks can respond to multiple memory requests at a time.

Each vault has a memory controller, or *vault controller*, in the logic die, and it is responsible for ensuring DRAM-specific timing constraints, DRAM sequencing, data routing across distributed vaults, error detection and correction, etc. The SerDes (Serialization/Deserialization) I/O interface enables high-speed communication between the host processor and a PNM stack, or between PNM stacks, with packetized abstract protocol.

In this study, we assume PNM cores are located in the logic die, one per vault; and a total of 16 PNM cores constitute a PNM processor.[1]

2.2 PNM system

Figure 1 shows a high-level overview of the PNM system, in which a host processor communicates with a PNM stack using high-speed I/O links. A PNM stack can be daisy-chained to other PNM stacks, or many PNM stacks can form either processor-centric networks or memory-centric networks to efficiently utilize the processor I/O bandwidth in a system [10].

An application can run on the host processor as in the traditional manner, or it can be offloaded to the PNM processor where data can be accessed in a more efficient way. In this work, we compare those two execution cases; one when a given application is executed on the powerful but power-hungry host multi-core processor and uses the PNM memory as a "dumb" memory, and the other when a given application is executed on the simple but power-efficient PNM processor where it can exploit enormous memory bandwidth without going through all the global wires associated with PCB traces on a motherboard. When a given application is executed on the PNM processor, the amount of data movement is expected to be small, but since it cannot have as deep a cache hierarchy as the host processor, the amount of DRAM array accesses might increase. It is important to understand the data movement dynamics in both execution scenarios so as to estimate the performance and energy aspects of exploiting computational capabilities of the PNM system.

3. ENERGY CHARACTERISTICS

In this section, we describe two execution scenarios for a given application, host execution and PNM execution, from the perspective of energy consumption. In particular, we focus on the data path that a memory request/response traverses through.

3.1 Host Execution

A host multi-core processor comprises multiple high performance, out-of-order cores, each of which contains its private cache(s). A large last-level cache (LLC) is shared across all cores within the processor and only LLC misses are serviced by the off-chip memory (here, "dumb" PNM memory). Every access to the off-chip memory navigates the global wires (PCB traces on a motherboard), and hence, consume energy associated with it.

Once the memory request reaches the PNM side, it is routed internally to the target vault. From this moment onward, a vault controller is fully responsible for handling the requests to the corresponding partition/bank/row. It schedules all the requests attempting to achieve the best performance, while conforming to the DRAM timing constraints.

A logic die and vertically stacked DRAM dies are connected using Through Silicon Via interconnects (TSVs). Data transfer through TSVs consumes less energy as compared to data transfer through PCB-based global wires. It is reported that the data transfer through TSVs is estimated to provide over 60x energy savings over the data transfer through global wires [8, 13].

Once a memory response has arrived in the memory controller in the logic die, it is sent back to the host processor, flowing through the interconnection network (in the form of switch) of the logic die that connects local vaults to the links, a link between the memory and the host processor, PCB-based global wires, and finally get installed in the cache.

3.2 PNM Execution

As described in Section 2.1, a PNM processor comprises a number of simple but power-efficient PNM cores. In our configuration we use an in-order core with a private single-level cache as a PNM core. It is worth noting that previous research focused on PNM designs which did not feature a cache hierarchy [6, 9]. However, recent studies include at least one level of cache hierarchy, as the idea that PNM would be expected to handle general purpose applications is gaining support [13, 22, 23].

When a given application is executed on the PNM processor, L1 misses are sent to the DRAM arrays through direct TSV connections. Memory accesses, as opposed to the host execution case, only navigate through TSVs, which consumes many orders of magnitude less energy as compared to the global wire traversals. According to [2, 8], each vault delivers 10 GB/s of memory bandwidth, provided by 32 TSV data lanes operating at faster than 1 GHz. Furthermore, if an application can be executed in parallel using 16 PNM cores, it can enjoy an order of magnitude more bandwidth (aggregate bandwidth of 160 GB/s) than current DDRx solutions while consuming less energy. On the other hand, since a PNM core cannot provide as much powerful computation capabilities as a host multi-core processor does, compute-intensive or sequential application would suffer from performance hit.

To improve system-wide throughput, the host processor can do other useful jobs while a given application is being executed in the PNM side: other threads of the same application or other programs. We leave such research, efficient system resource utilization, for the future since the primary objective of this work is to understand energy consumption characteristics of the PNM architecture.

3.3 Energy Consumption Breakdown

We have constructed an analytical performance and energy model to roughly estimate how much energy will be consumed when an application is executed on the host processor and when it is executed on the PNM processor, a collection of PNM cores. For this, the architectural differences between the host processor and the PNM processor are considered. In addition, the energy consumption by the medium that each data traverses through is also taken into account. Before we present our analytical model, which will be discussed in Section 4, we provide an energy consumption breakdown of a processor, including energy consumed at each operational stage, energy consumed for the caches as well as the energy consumed for un-core part.

3.3.1 *Host Processor Side*

Throughout the execution, a core consumes different pow-

[1]We use 16 vaults organization based on [15], in which 16 vaults organization is reported to achieve the highest output bandwidth with 256 total banks.

ers at different stages. Intel Xeon Processor E3-1275, for example, is reported to dissipate 95W when operating at base frequency with all cores active [3], while dissipating only 23.6W with all cores idle [1]. Intel Core/Core 2 Duo is reported to consume 0.5W to 1.05W on average when idle [12] depending on the processor type and the frequency of active/idle state transition.

We split core's power consumption into three categories: logic, cache, and miscellaneous. We assume each host core, excluding on-chip caches, consumes 10W active power and 1W idle power; we approximate each core's idle power at 10% of its active power. We split cache's power consumption into static and dynamic, so that application's memory access characteristics can be reflected in the estimation of overall energy consumption. Lastly, we categorize power consumption due to other logics as miscellaneous, since they cannot be easily powered down for their wake-up latency having relatively high impact on performance.

3.3.2 PNM Processor Side

It is reported that energy per bit for HMC access is measured at 10.48 pJ, of which about 3.7 pJ is consumed in the DRAM dies and about 6.78 pJ is consumed in the logic die [8]. According to [8, 17], about 43% of total logic die power is consumed in the SerDes circuits for high-speed communication. Note that power gating technique cannot be easily applied to the SerDes circuits because of their relatively long wake-up latency.

We assume that each PNM core, excluding on-chip caches, consumes 80mW active power and 8mW idle power: a total of 1.28W will be consumed if all PNM cores are in active mode [13]; again, we approximate each core's idle power at 10% of its active power. We assume each PNM core has a single-level L1 cache, and this will be reflected in the estimation of overall energy consumption.

3.3.3 Global Transfer

It is reported that the global data transfer through PCB consumes more than 60x energy than the data transfer through TSVs [8, 13].[2] More details on our analytical model will be discussed in Section 4.

4. ANALYTICAL ENERGY MODEL

Our analytical model takes application characteristics and estimates the energy consumption on both host and PNM execution cases. Application characteristics include the number of cache accesses for each level, the number of memory accesses, and the number of cycle during which the core is in active or idle state. We profile the target application for both host and PNM execution to collect such information.

Table 1: Power and energy parameters of each component

SRAM Leakage Power	4.050 nW	per cell	[7, 11]
L1 Cache Access Energy	0.494 nJ	per access	[7, 11]
L2 Cache Access Energy	3.307 nJ	per access	[7, 11]
L3 Cache Access Energy	6.995 nJ	per access	[7, 11]
DRAM Background Power	0.470 W	per package	[13]
DRAM Access Energy	28.034 nJ	per access	[20]
TSV Transfer Energy	0.078 pJ	per bit	[19, 21]
Global Transfer Energy	4.700 pJ	per bit	[13]

[2]A 3D TSV traversal is expected to consume on the order of 30-110 fJ/b.

Power and energy parameters for our analytical model are summarized in Table 1. Please note that these parameters are dependent on the process technology generations, design choices, the size of the component, as such. The energy consumed for each cache access depends on the process technology as well as the size of the cache, for which we used the value modeled from CACTI [11]. The energy consumed for each DRAM access used in this study is borrowed from [20], with which we summed up the energy for the decoder, wordline, senseamps, bitlines, and termination resistors. The global transfer energy (PCB) and TSV transfer energy value is borrowed from [13] and [19, 21], respectively. Please note that the global transfer energy is consumed for data movement between the host processor and the PNM to navigate the global wires. Therefore the global transfer energy is highly dependent on the distance between both ends, and one would need different value for their own floorplan.

Our analytical model reports back the estimated energy consumption for the given application on both execution cases. Note that one can profile any piece of code, such as function, loop or any region of interest and feed the profiled data into the analytical model to get a high-level sense whether it is suitable for PNM execution.

The following factors are taken into account to estimate overall energy consumption.

- Processor energy (Core & Un-core portions)
- Cache energy (Leakage & Access)
- DRAM energy (Background & Access)
- Data movement energy

Total energy consumption is a summation of the energy consumed by host processor and memory parts, and the energy consumption due to the data movement.

$$E_{TOTAL} = E_{HOST} + E_{PNM} + E_{GLOBAL\_XFER} \tag{1}$$

Host energy consumption can be expanded to:

$$E_{HOST} = E_{HOST\_LOGIC} + E_{HOST\_CACHE} \tag{2}$$

The host processor's logic component energy consumption can be attributed to either core or un-core. We assume the un-core portion of the processor cannot be power-down because of the long wake-up latency.

$$E_{HOST\_LOGIC} = E_{CORE} + E_{UNCORE} \tag{3}$$

$$E_{CORE} = P_{ACTIVE} \times \sum^{N} t_{ACTIVE} + P_{IDLE} \times \sum^{N} t_{IDLE} \tag{4}$$

$$E_{UNCORE} = N_{CHANNEL} \times P_{UNCORE} \times t_{TOTAL} \tag{5}$$

Cache energy consumption can be divided into either static or dynamic. Static and dynamic energy are proportional to the size of the cache and the number of accesses, respectively.

$$E_{HOST\_CACHE} = E_{STATIC} + E_{DYNAMIC} \tag{6}$$

[2]Different cache access energy value is used for each cache level.

$$E_{STATIC} = \sum^{L} P_{SRAM\_LEAKAGE} \times t_{TOTAL} \times N_{SRAM} \quad (7)$$

$$E_{DYNAMIC} = \sum^{L} E_{ACCESS} \times N_{ACCESS} \quad (8)$$

PNM's energy consumption can be attributed to either logic or memory.

$$E_{PNM} = E_{PNM\_LOGIC} + E_{PNM\_MEMORY} \quad (9)$$

As in the host processor, logic component energy consumption comes from either core or un-core portion.

$$E_{PNM\_LOGIC} = E_{CORE} + E_{UNCORE} \quad (10)$$

The same equation can be applied to the PNM core, though actual power consumption would be different.

$$E_{UNCORE} = (N_{LINK} \times P_{LINK} + P_{MISC}) \times t_{TOTAL}{}^{3} \quad (11)$$

It is worth noting that in the host execution case, we assume $E_{PNM\_LOGIC} = E_{UNCORE}$; PNM cores are turned off. Memory energy consumption can be divided into static and dynamic. Note TSV data transfer energy is included in the dynamic portion of memory energy consumption.

We assume an equal amount of energy to be consumed regardless of whether the requested data is present in an open row (a row buffer hit) or not (a row buffer miss). To improve the accuracy of our model, the memory access energy to an open row has to be differentiated from the energy to a closed row. We leave this potential improvement for future work.

$$E_{PNM\_MEMORY} = E_{STATIC} + E_{DYNAMIC} \quad (12)$$

$$E_{STATIC} = P_{DRAM\_BACKGROUND} \times t_{TOTAL} \quad (13)$$

$$E_{DYNAMIC} = (E_{ACCESS} + E_{TSV\_XFER}) \times N_{ACCESS} \quad (14)$$

Global transfer energy accounts for the energy consumed by off-chip data transfer through PCB-based bus.

$$E_{GLOBAL\_XFER} = E_{PCB\_XFER} \times N_{DRAM\_ACCESS} \quad (15)$$

5. METHODOLOGY

We conduct preliminary experiments to estimate the performance and energy efficiency of PNM execution with our analytical energy model in conjunction with cycle-level architecture simulation frameworks. We use The Structural Simulation Toolkit (SST) [14] with MacSim [4], a cycle-level heterogeneous architecture simulator. We use DRAMSim2 [16] for DRAM timing simulation, which has been modified to model an HMC-based PNM architecture. For detailed HMC model, we use the timing parameters used in [15].

Our evaluation is performed in two steps; first, detailed performance characterization is done with simulators. Next, overall system energy is estimated with analytic energy model. Specifically, we use cycle-level simulators to collect the performance numbers, such as the number of cache accesses for each level, the number of memory accesses, and each core's active/idle cycle counts for each application in both host and PNM executions. With these, we estimate the energy consumption on both cases using our analytical energy model.

5.1 Configuration

We model a high performance host processor with 4 GHz, quad-cores, in which each core executes instructions out-of-order with 256-entry ROB. Each core has a private L1 and L2 caches, and a last-level L3 cache is shared across all cores. The architecture details and power consumptions per operating mode for the host processor are summarized in Table 2.

Table 2: Architecture & Powers for the host Processor

Processor	Quad core, 4GHz
Core Parameter	4-wide OOO, 256-entry ROB
Cache Linesize	64 Byte
L1 Cache	32KB I and D, 3 cycle
L2 Cache	128KB, 8 cycle
L3 Cache	2MB, 30 cycles
# Channels	4
Memory Latency	L1,L2,L3 + Interconnect + PCB/TSV + DRAM
P_{ACTIVE}	10.000W
P_{IDLE}	1.000W
P_{UNCORE}	10.000W
E_{PCB_XFER}	4.700 pJ/bit
E_{TSV_XFER}	0.078 pJ/bit

The PNM processor is modeled to be a collection of 1 GHz, in-order PNM cores, each of which has a single-level instruction and data cache. The architecture details and power consumptions per operating mode for the PNM processor are summarized in Table 3.

Table 3: Architecture & Powers for the PNM Processor

Processor	16 core, 1GHz
Core Parameter	Single-issue, in-order
Cache Linesize	64 Byte
L1 Cache	32KB I and D, 3 cycle
# Links	4
# Partitions	16
# Banks per Partition	2
Memory Latency	L1 + TSV + DRAM
P_{ACTIVE}	80mW
P_{IDLE}	8mW
P_{UNCORE_SERDES}	1.445W per link
P_{UNCORE_MISC}	2.890W per package
E_{TSV_XFER}	0.078 pJ/bit

We use the timing parameters used by [15] for 3D-stacked memory model. Specifically, we use $t_{RP} = 17$, $t_{CCD} = 6$, $t_{RCD} = 17$, $t_{CL} = 17$, $t_{WR} = 19$, $t_{RAS} = 34$ cycles for DRAM timing parameters.

5.2 Benchmarks

We use a representative slice [18] of 200M instructions for 11 HPC mini-applications from Mantevo benchmark suite

[3] P_{LINK} and P_{MISC} refer to P_{UNCORE_SERDES} and P_{UNCORE_MISC}, respectively, from Table 3

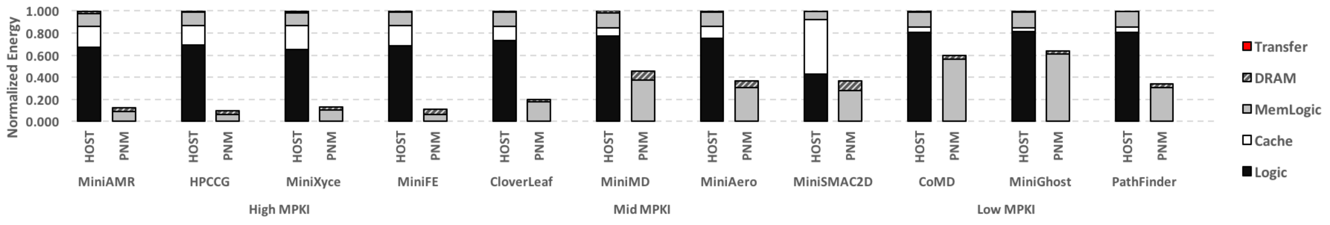

Figure 2: Energy consumption of host and PNM execution cases, normalized to that of host execution case. Note that we assume PNM cores are turned off in the host execution. Hence, we use Eq 10 for the PNM execution and Eq 11 for the host execution for the energy consumed in the logic die.

[5]. Table 4 lists the applications used in our study and their characteristics - the misses per kilo instructions (MPKI) of last-level cache (LLC) and instruction-level parallelism (ILP). We categorize applications into three different classes based on LLC MPKI. Those with average MPKI rates greater than 25 fall into **High MPKI** and those with average MPKI rates less than 1 fall into **Low MPKI**. The remaining applications are grouped as **Mid MPKI**.

Table 4: List of Mantevo mini-applications [5] and each application's misses per kilo instructions of last-level cache (LLC MPKI) and instruction-level parallelism (ILP). These are measured from host execution.

Class	Benchmark	MPKI	ILP
High MPKI	MiniAMR	60.915	1.369
	HPCCG	35.914	1.111
	MiniXyce	34.032	1.467
	MiniFE	28.383	1.104
Mid MPKI	CloverLeaf	6.924	1.431
	MiniMD	4.858	3.082
	MiniAero	3.203	1.867
Low MPKI	MiniSMAC2D	0.642	1.399
	CoMD	0.352	3.556
	MiniGhost	0.105	3.864
	PathFinder	0.093	1.745

6. RESULTS AND ANALYSIS

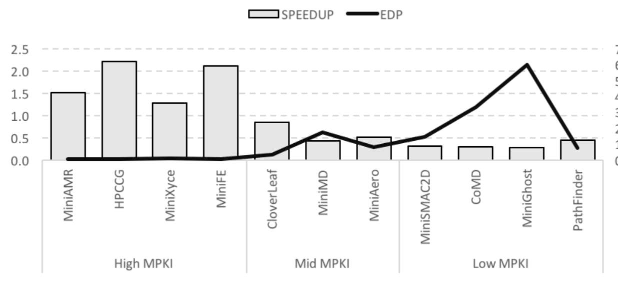

Figure 3: Speedup of PNM execution and Energy-Delay Product for each benchmark, normalized to the host execution case.

Figure 2 shows the energy consumption of each application for both host and PNM execution cases. The energy consumption is normalized to the host execution case. Each bar further details the energy consumed by each architectural components. Note that the host processor is assumed to do other useful jobs to improve system-wide throughput while a given application is being executed in the PNM side. In other words, we do not include the host processor's static energy consumption in the PNM execution case.

The energy savings are realized across all evaluated benchmarks with PNM execution. These energy savings are obtained for mainly two reasons: power-efficient core and shallow cache hierarchy, thereby avoiding exess memory access latency. For benchmarks that have high MPKI, PNM execution outperforms the host execution counterpart by an average of 73.3% while an average of 88.30% energy savings are obtained. Benchmarks that have mid or low MPKI, however, experience average performance degradations of 43.7% and 67.6%, respectively, with PNM execution, while modest energy savings (67.85% and 53.17% reduction), yet still significant, are achieved. Thus, for these benchmarks, a large performance degradation must be tolerated in order to obtain energy savings. It is worth noting that performance degradation does not necessarily mean energy inefficiency. As shown in the bar chart in Figure 3, *CloverLeaf* experiences performance degradation of 18.8% while about 80.0% energy savings are obtained.

Therefore, we infer that one important application characteristic to realize a significant savings in energy consumption without performance degradation is high LLC MPKI. Line chart in Figure 3 shows the Energy-Delay Product (EDP) for each benchmark, normalized to the host execution case. It is clearly shown that high MPKI benchmarks are PNM-friendly in terms of both performance and energy consumption. Average memory access latency of *MiniFE* being reduced by 52.7% when the application is executed on PNM processor also supports this argument. Other high MPKI applications show similar behavior.

We also observed interesting results for benchmarks that have similar MPKIs; *CloverLeaf* and *MiniMD*. While 80.0% energy savings are obtained with 18.8% performance degradation for the former, the latter experiences about 137.6% degraded performance for 54.5% energy savings. These aspects are clearly shown in Figure 3; the line chart shoots up when moving from *CloverLeaf* to *MiniMD*. This discrepancy is due to different level of instruction-level parallelism (ILP) each application has; specifically, the performance boost for *MiniMD* when out-of-order core is used is 167.3%, while that for *CloverLeaf* is more modest; 39.3%. This explains the reason as to why *MiniMD* sees more performance degradation than *CloverLeaf* does, even though they have similar

MPKIs. Furthermore, benchmarks that have high ILP, for instance, *MiniMD*, *CoMD*, *MiniGhost*, experience particularity high performance loss when executed on PNM processor. Hence, relatively low energy reductions are achieved.

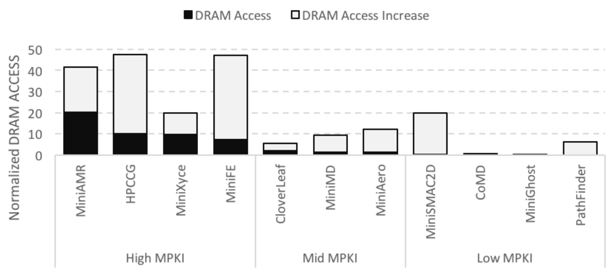

Figure 4: The number of DRAM accesses for host execution case, and its relative increment for PNM execution case.

Figure 4 shows the number of DRAM accesses for both execution cases, in which the number is normalized to the average number of DRAM accesses across all benchmarks. This clearly shows that the more dram accesses an application has, the more PNM-friendly the application is.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we analyze the energy consumption characteristics of a processing-near-memory (PNM) system that has a conventional processor and a 3D memory stack with fully-programmable cores. We provide comprehensive expositions on the data movement dynamics in such a system when computations are performed in the host processor and when the computations are performed in the PNM processor. Based on the underlying architecture and the technology with which each component is built, we construct a high-level analytical energy model. From the preliminary experiments with 11 HPC benchmarks from Mantevo benchmark suite, we observed that LLC MPKI is one important characteristic in determining the friendliness of the application to the PNM execution; High MPKI benchmarks enjoy both performance and energy benefit from PNM execution.

Although this study focused on a HMC-based PNM architecture, it can easily be extended to other memory technologies, such as High Bandwidth Memory (HBM). Furthermore, since the concept of PNM is not necessarily restricted to the 3D-stacking memory, another form of PNM architecture is also conceivable. Exploring these extensions is a part of our future work.

8. ACKNOWLEDGMENTS

We gratefully acknowledge the support of National Science Foundation (NSF) XPS-1337177 and Sandia National Laboratories. We would like to thank the anonymous reviewers for their comments and suggestions. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF or Sandia National Laboratories.

9. REFERENCES

[1] Haswell-Based Xeon E3-1200. http://goo.gl/EDF3nh.

[2] Inside the HMC. http://goo.gl/DYoMY4.

[3] Intel Xeon Processor E3-1275. http://goo.gl/EjmNJd.

[4] MacSim Simulator. https://goo.gl/gkosY6.

[5] Mantevo. https://mantevo.org/.

[6] D. Elliott, M. Stumm, W. Snelgrove, C. Cojocaru, and R. McKenzie. Computational RAM: implementing processors in memory. 16(1):32–41, Jan 1999.

[7] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 148–157, 2002.

[8] J. Jeddeloh and B. Keeth. Hybrid memory cube new DRAM architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*, pages 87–88, June 2012.

[9] 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 *Computer Design, 1999. (ICCD '99) International Conference on*, pages 192–201, 1999.

[10] G. Kim, J. Kim, J. H. Ahn, and J. Kim. Memory-centric system interconnect design with hybrid memory cubes. In *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pages 145–155, Sept 2013.

[11] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A Tool to Understand Large Caches, 2009.

[12] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the Intel core duo processor. *Intel Technology Journal*, 10(2), 2006.

[13] 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 *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 190–200, March 2014.

[14] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, Mar. 2011.

[15] P. Rosenfeld. *Performance Exploration of the Hybrid Memory Cube*. Ph.D. dissertation, University of Maryland, College Park, 2014.

[16] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Comput. Archit. Lett.*, 10(1):16–19, Jan. 2011.

[17] G. Sandhu. DRAM Scaling & Bandwidth Challenges. In *NSF Workshop on Emerging Technologies for Interconnects (WETI)*, February 2012.

[18] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pages 45–57, New York, NY, USA, 2002. ACM.

[19] A. N. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi. Combining Memory and a Controller with Photonics Through 3D-stacking to Enable Scalable and Energy-efficient Systems. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 425–436, New York, NY, USA, 2011. ACM.

[20] 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 *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 175–186, New York, NY, USA, 2010. ACM.

[21] 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 *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12, Jan 2010.

[22] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski. TOP-PIM: Throughput-oriented Programmable Processing in Memory. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 85–98, New York, NY, USA, 2014. ACM.

[23] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. Greathouse, M. Meswani, M. Nutter, and M. Ignatowski. A New Perspective on Processing-in-memory Architecture Design. In *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, MSPC '13, pages 7:1–7:3, New York, NY, USA, 2013. ACM.