Exploring Non-Uniform Processing In-Memory Architectures

Kishore Punniyamurthy
kishore.punniyamurthy@utexas.edu
The University of Texas at Austin

Andreas Gerstlauer
gerstl@ece.utexas.edu
The University of Texas at Austin

ABSTRACT

Advancements in packaging technology have made in-package high- bandwidth 3D memory a reality. Traditional PIM approaches have explored exploiting the higher bandwidth but limited amount of compute available in external 3D memories to reap performance benefits. By contrast, the feasibility of having memories in the same package as the centralized compute units has made novel processing in-memory architectures viable within exascale nodes. A single such node architecture can range from a centralized compute element with in-package memory connected to an external memory network to multiple processing-in-memory elements with uniform or non-uniform memory-to-compute ratios. However, the optimal distribution of compute and memory for different applications remains an open question to be evaluated. Distributing applications across multiple compute elements could increase the off-chip accesses but may reduce contention. This paper evaluates different non-uniform processing in-memory (NUPIM) architectures for GPGPU applications. We identify and discuss the factors that influence architecture choices for a given application. We observe that for memory-intensive irregular benchmarks, non-uniformly distributed processing in memory architectures perform better than centralized compute even though the former causes higher off-chip accesses.

1 INTRODUCTION

The growing demand for computational capability in scientific and high-performance computing have made exascale systems a necessity [20]. In such large-scale systems, data-movement is a significant concern [7]. The emergence of 3D stacked memory has provided opportunities to place compute units within the logic layer of the memory stack. These in-memory compute units have greater bandwidth provided by the through-silicon vias (TSVs) between stacked logic and memory layers. Recent work has shown that processing-in-memory (PIM) architectures, which place compute units in external memory stacks (Fig. 1) can potentially provide performance benefits owing to the higher memory bandwidth available compared to the central compute unit. Placing computations directly into external memory stacks can reduce or eliminate data movement across chips. However, traditional PIM approaches have been limited in the amount of compute capabilities that can be realized in logic layers of existing memory stacks.

Improvements in die-stacking capabilities are projected to allow us to integrate more compute and high-bandwidth 3D stack memory within the same package. Such advancements are expected to be widely used to meet the bandwidth requirements of an exascale node [22]. The feasibility of such technology opens up many new possibilities for future exascale node architectures. However, it also introduces new types of design questions and tradeoffs. Assuming an equal amount of total compute capability, a node could be architected as a single large compute element with in-package memory, connected to an external memory network (to meet overall memory capacity requirements) as shown in Fig. 2a. Or it could have multiple compute elements with non-uniform (Fig. 2b) or uniform (Fig. 2c) memory-to-compute ratios. In general, exascale nodes can have a non-uniform processing-in-memory (NUPIM) architecture consisting of multiple packages of varying sizes each having different proportion of memory and compute integrated together. However, evaluating the different architecture options and their application-specific benefits and tradeoffs is not trivial.

For irregular applications with large amounts of data-dependent accesses and page-sharing, a centralized compute (Fig. 2a) architecture might appear as suitable choice since it would result in minimum off-chip accesses. However, the impact of contentions in the central compute element can negate any performance improvement. Architectures with multiple compute elements might help reduce contention but could potentially result in a higher number of remote accesses. Depending on the application's sensitivity towards off-chip accesses and contentions, some architectures will perform better than others. However, if the application is regular and can be partitioned, the effect of contention and off-chip accesses are no longer conflicting. There are many factors that need to be considered when choosing a suitable architecture for a range of applications.

In this paper, we evaluate different architectures and study the factors that determine the suitability of an architecture for a given application. Our evaluation uses GPGPU applications and Streaming Multiprocessors (SMs) as our compute units. However, the insights obtained from this work will apply for NUPIM architectures within exascale nodes with arbitrary types of compute units (CPUs, GPUs, accelerators).

The rest of the paper is organized as follows: Section 2 provides an analysis of memory access patterns of applications and discusses the suitability of architectures based on the same. Section 3 presents the various architectural possibilities that are expected to be feasible by the anticipated time-frame of exascale systems. Our experimental setup and results are presented in Section 4. The paper concludes with a summary and an outlook on future work in Section 6 after discussing the related work in Section 5.

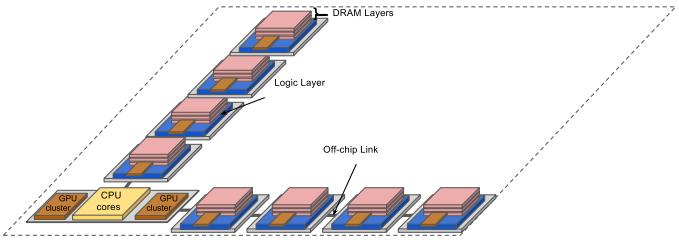
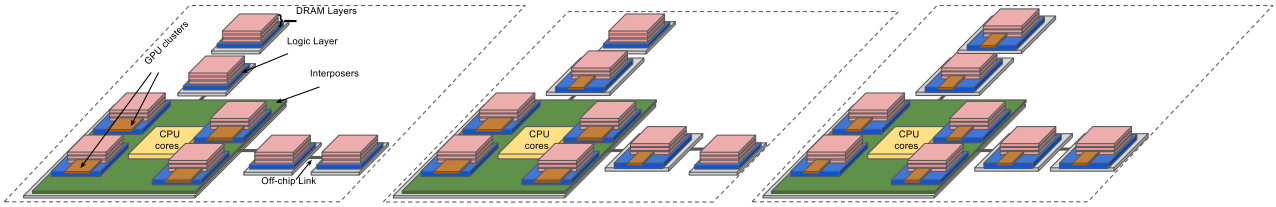


Figure 1: PIM Architecture.



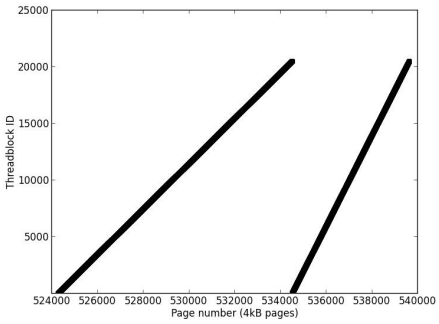
(a) Centralized compute. **(b) Non-uniform multi-element.** **(c) Uniform multi-element.**

Figure 2: Potential exascale node architectures.

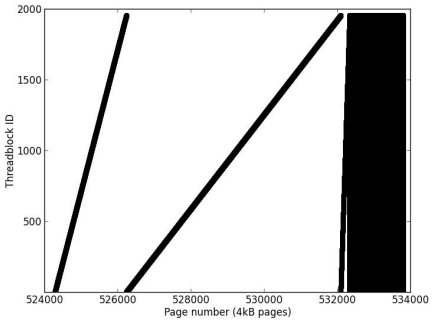
2 APPLICATION ANALYSIS

In order to study and understand the suitability of an architecture for any application, it is necessary to understand the memory access pattern of the application. Applications with different inter-thread sharing behavior will perform differently across different architectures. To study the memory access pattern, we plot in Fig. 10 the IDs of threadblocks (Y-axis) launched in a kernel against the page (4kB each) numbers accessed by the threadblocks. We classify the benchmarks into two categories: low and high sharing based on the pre-dominant behavior exhibited by the benchmarks. In reality, different kernels within the same benchmark can exhibit different behaviors as shown later. Fig. 3a and Fig. 3b show the behavior of two representative benchmarks from each category.

Fig. 3a shows the memory access pattern of the k-Nearest Neighbour (NN) benchmark. The plot shows that pages upto 534k are regularly accessed by the threadblocks starting at ID 0, at which point they wrap around and subsequent pages are accessed again starting from threadblock ID 0. There are 2 major observations to note from the graph. Firstly, the benchmark has high locality i.e. there is not much page sharing among threadblocks. Secondly, memory footprint scales with the number of threadblocks. Such patterns are typical due to the regular nature of the benchmark. Since



(a) NN



(b) BFS (Z6Kernel)

Figure 3: Benchmark memory access patterns.

the memory footprint scales with threadblocks, the working set of such applications will eventually spill to external memory in a centralized compute architecture. However, lack of inter-threadblock sharing allows the threadblocks and its corresponding working set to be distributed among multiple elements, without increasing remote accesses significantly.

Fig. 3b shows the access pattern of the *Z6kernel* from the BFS benchmark. As can be seen from the graph, pages from 524k to 532k are accessed regularly by the threadblocks, wrapping around at 526k. The pages from 523k to 534k are accessed irregularly by the threadblocks. To better understand the extent of page sharing in the lower versus upper memory regions in this benchmark, we further plot the benchmark's communication matrix (Fig. 4). A Communication matrix is a NxN matrix in which each element (i,j) represent the total number of shared page accesses made by threadblocks i and j. Here, N is the total number of threadblocks launched. In our representation, the darker the element in the communication matrix, the higher the amount of page sharing. Fig. 4 shows the communication matrix for BFS (Z6kernel). We see that there is significant page sharing among different threadblocks. Such patterns result from the abundance of data-dependent accesses. For such applications, distributing threadblocks and their corresponding working set among multiple elements is difficult and usually results in increased remote accesses. An architecture with centralized compute might be better since it will result in minimum remote accesses.

3 ARCHITECTURES EVALUATED

Previous works have shown that in-package high-bandwidth memory will be essential for an exascale node, but that external memory will also be required to meet the capacity demands [22]. Under the assumption of equal amounts of total memory and compute per node, all SMs can be centrally placed, or distributed among external memory stacks in uniform or non-uniform manner. Based on this, we evaluate the following 3 types of architectures (Fig. 5):

(1) *Centralized compute* (Fig. 5a): All the SMs are placed in a central package with in-package high bandwidth memory and an external memory network.

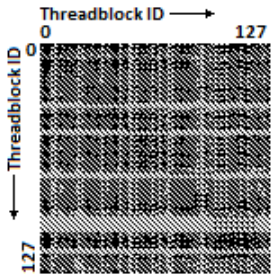
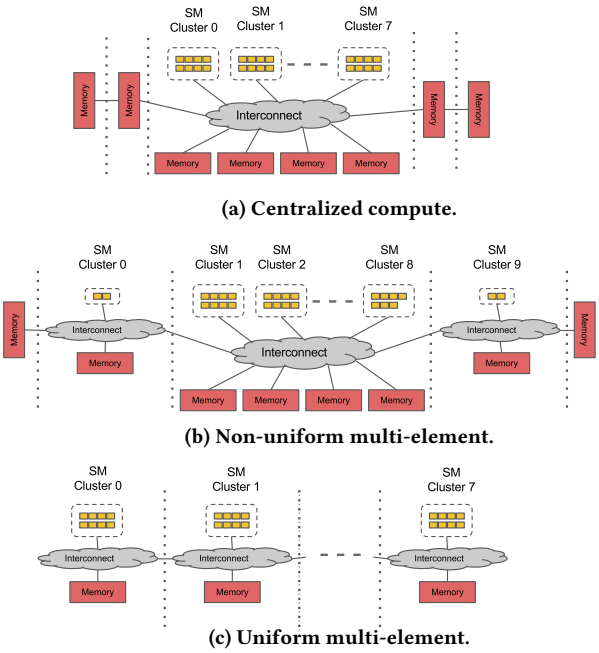


Figure 4: BFS communication matrix.

(2) *Non-uniform multi-element* (Fig. 5b): Multiple processing-in-memory elements are interconnected together. The elements have different memory-to-compute ratios. We specifically evaluate the following two configurations combining a large centralized compute with in-package memory (big) connected to external memory stacks with no or small number of in-memory SMs in their logic layer (little):
   - *BigLittle2*: This configuration has in-memory SMs in both the memory stacks at the first level of the memory network, but no compute in levels beyond.
   - *BigLittle4*: This configuration has in-memory SMs in 4 memory stacks present at the first and second level of the memory network.

(3) *Uniform multi-element* (Fig. 5c): Multiple elements having identical memory-to-compute ratio are interconnected together. In order to show an extreme case of remote accesses, we envision this architecture as uniform memory stacks with compute interconnected with each other instead of a large centralized package with memory connected to external stacks (all having the same memory-to-compute ratio).

Data mapping significantly affects the performance of the application. Previous work [18] suggested that a locality-based data mapping suits regular benchmarks with low page sharing. Hence, while evaluating benchmarks with low page sharing, we map data based on the locality policy as presented in [18]. We use memory traces to statically place the pages into appropriate memory stacks. Other work has demonstrated that interleaving data provides best performance for irregular benchmarks with high page sharing [8]. For such benchmarks, we interleave data across controllers in chunks of 256 Bytes in our evaluation.



(a) Centralized compute.

(b) Non-uniform multi-element.

(c) Uniform multi-element.

Figure 5: Node architectures evaluated.

4 EXPERIMENTS AND RESULTS

The exact simulator parameters for different configurations evaluated are shown in Table. 1. We use GPGPUSim v3.2.2 [6] for our experiments. The simulator has been modified to replicate our different configurations. We evaluate different architectures using memory-intensive benchmarks from Rodinia [21], Parboil [13] and Lulesh [11] benchmark suites, including four benchmarks in the high (*BFS, Lulesh, Btree, Spmv*) and one benchmark in the low (*NN*) sharing categories. The list of benchmarks used in our evaluation is provided in Table. 2.

In order to keep simulation time reasonable, we simulate a smaller configuration compared to a typical exascale node configuration. We model 3D stack memory as Hybrid Memory Cube (HMC) [5], since it allows integration of units into a network [22]. The capacity of each stack is the same and total memory is equal to the memory footprint of the application evaluated. We assume up to 2 GTX480 SMs fit within the logic layer of memory stacks. Considering the area of Hybrid Memory Cube (HMC) [26] and vault controllers [3], our calculations show that 2 GTX480 SMs [1] can comfortably fit within the logic layer.

4.1 Performance Results

Fig. 6 shows the speedup of benchmarks executed in different architecture configurations normalized to the centralized compute architecture. Fig. 7 and Fig. 8 present the number of cache lines moved over the off-chip link (data movements) and the number of memory stalls due to interconnect congestion, both normalized to central compute results, for different architectures.

As expected, benchmarks with low sharing (*NN*), perform better (1.78x) in a uniform multi-element architecture than a centralized compute. The threadblocks and its corresponding working set can be partitioned owing to the strong locality and weak sharing demonstrated by these benchmarks. This is evident from the lower remote accesses and interconnect stalls compared to centralized compute. However, the actual speedup while using a practical thread and data mapping policy is expected to be less.

For benchmarks with high sharing (*BFS, SPMV, Btree, Lulesh*), we see that *BigLittle2* and *BigLittle4* architectures perform better than both centralized compute and uniform multi-element architectures. *BigLittle2* has an average speedup of 1.06x (up to 1.10x), while *BigLittle4* has an average speedup of 1.13x (up to 1.19x) over *centralized_compute*. As expected from studying their memory access patterns, distributing these applications among multiple elements results in higher remote accesses. *BigLittle2* has an average 1.07x higher off-chip hops compared to centralized compute, while *BigLittle4* and uniform multi-element architectures have 1.20x and 3.47x higher average off-chip hops. However, the reduction in interconnect contention offsets the adverse performance impact caused by the higher remote accesses. *BigLittle2* has 0.91x lower interconnect stalls due to congestion on average, while *BigLittle4* and uniform multi-element architectures have 0.83x and 0.98x lower average interconnect stalls. The reduction in contention dominates the effect of higher off-chip accesses for *BigLittle2* and *BigLittle4* architectures. However, off-chip accesses for a uniform multi-element architecture are too large to be offset by reductions in contention, resulting in performance degradation compared to centralized compute.

Table 1: Simulation parameters.

Architectures		
Centralized compute	# SMs	64
	# Clusters	8
	In-pkg DRAM stacks	4
	DRAM stacks w/o compute	4
Multi-unit homogeneous	# SMs	64
	# Units	8
	Clusters/unit	1
	In-pkg DRAM stacks/unit	1
Multi-unit heterogeneous (*bigLittle2,bigLittle4*)	# Total SMs	64
	SMs/Little unit	2
	Clusters in big	8
	Clusters/Little unit	1
	In-pkg DRAM stacks in big	4
	DRAM stacks/Little	1
bigLittle2	# SMs in big	60
	# Little units	2
	DRAM stacks w/o compute	2
bigLittle4	# SMs in big	56
	# Little units	4
	DRAM stacks w/o compute	0
SM Configuration		
Core configuration	1.4Ghz, 48 warps/SM, 32 threads/warp, 32768 register, GTO warp scheduler, 8 CTAs/SM, 48kB shared memory [2]	
Private L1 cache	16kB, 4-way, 128B block size[2]	
Shared L2 cache	768kB, 8-way,128B block size [2]	
Bandwidth		
Interconnect	Crossbar, flit size=32B [2]	
In-pkg bandwidth	231 GB/s per stack	
Off-chip bandwidth	115.5 GB/s	
Memory Stack		
Memory stack configuration	8 memory stacks, 16 vaults/stack, 16 banks/vault, 64 TSVs/vault [15]	
DRAM Scheduling policy	FR-FCFS	
DRAM Timing	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{CCD}=2, t_{RCD}=12, t_{RRD}=6, t_{CDLR}=5, t_{WR}=12 [2]	

Table 2: Benchmarks evaluated.

Benchmark	Page sharing	Input
NN	Low	5120k
BFS	High	Graph1MW
Lulesh	High	Default (first 26 kernels)
SPMV	High	Large (first 5 kernels)
Btree	High	Default (1M)

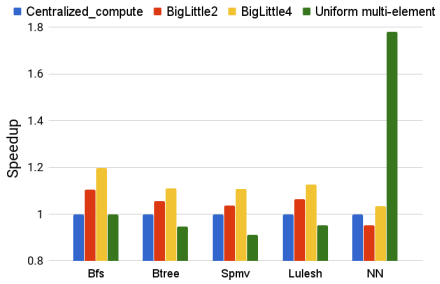


Figure 6: Performance speedup over centralized compute.

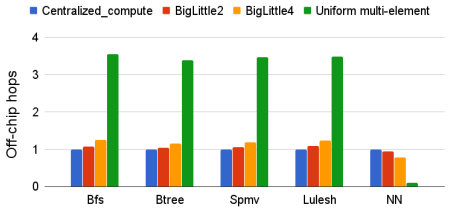


Figure 7: Off-chip hops.

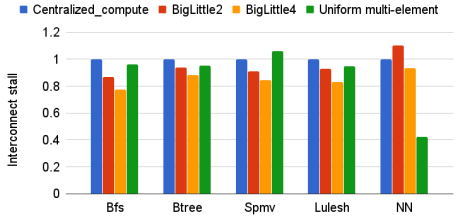


Figure 8: Interconnect congestion.

4.2 Deepdive: Lulesh

We take a closer look at the speedup obtained on each architecture while executing the *Lulesh* benchmark. The speedup for each *Lulesh* kernel evaluated is shown in Fig. 9. Not surprisingly, not all kernels get similar speedup for the *BigLittle4* architecture. For example, *CalcPressureForElems* and *UpdateVolumeForElems* kernels perform equally good in a centralized compute architecture as compared to *BigLittle4*. Furthermore, it can be seen that *BigLittle2* performs equally good for *CalcLagrangeElementsPart2* and *CalcPressureForElems* kernels, and it outperforms *BigLittle4* in the *ApplyAccelerationBoundaryCondition* kernel. The reason is less contention. Fig. 10a shows the memory access pattern for *IntegrateStressForElems* kernel of *Lulesh*. The pages are accessed irregularly by the threadblocks, causing high contention and thereby favouring *BigLittle4* architecture over others. However, the pages are accessed regularly in *CalcLagrangeElementsPart2* kernel as shown in Fig. 10b, resulting in less contention. For such kernels, a *BigLittle4* architecture is not suitable because even if the performance appears similar, off-chip accesses are higher in *BigLittle4* compared to *BigLittle2* and centralized compute architectures. Recent studies have shown memory I/O as major contributor to power consumption [23]. As such, executing the above benchmarks in a *BigLittle4* architecture can be expected to not only result in no performance gain, but also higher power consumption. Fig. 7 shows that cumulative off-chip accesses for *Lulesh* are 1.24x higher than for centralized compute. Out of this increase in off-chip accesses, 23% are due to kernels providing less than 10% performance speedup, indicating that there

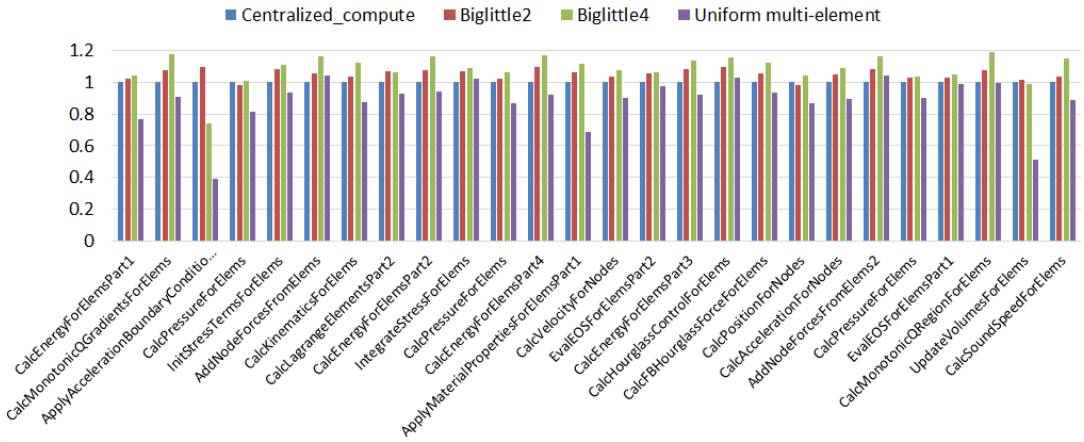


Figure 9: Speedup observed in different kernels of the Lulesh benchmark.

is a potential for power savings and performance improvements by dynamically selecting suitable architectures for each kernel in corresponding runtime systems.

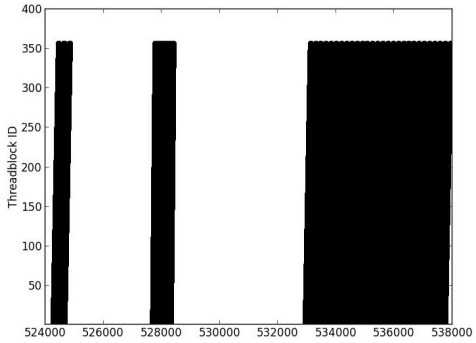
5 RELATED WORK

This paper evaluates and compares the performance of different non-uniform processing-in-memory (NUPIM) architectures for future exascale nodes enabled by advancements in packaging technology. To the best of our knowledge, no prior work has studied such NUPIM architectures for exascale nodes. In the following, we discuss related work in processing-in-memory (PIM), non-uniform memory access (NUMA) and heterogeneous memory system architectures.

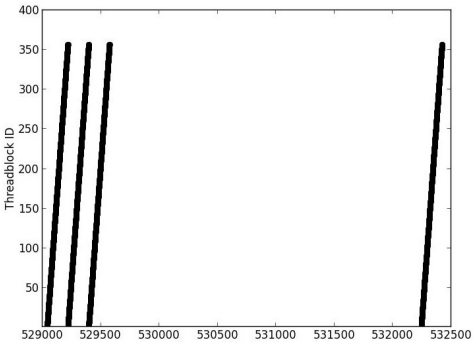
The idea of processing-in-memory has been around for long [25, 27, 30, 32] and has gained renewed importance in recent years with the emergence of 3D stacked memory technology. Many previous works have proposed deriving performance improvements and power savings for memory-intensive applications by using an accelerator or compute unit integrated with the 3D stack memory [10, 12, 24]. This paper differs from previous PIM works in following ways:

(1) Previous works on PIM suggest offloading memory-intensive parts of computations such that the net off-chip data movement is reduced [15]. With modern packaging technology allowing in-package high-bandwidth 3D stack memory, offloading compute to external in-memory compute units can, by contrast, result in higher remote accesses. This paper studies the effects of distributing computations across multiple heterogeneous and non-uniform processing-in-memory units. Our experiments demonstrate that, due to reduced contentions in centralized units, there are benefits of distributing even if it results in higher remote accesses. We study corresponding factors that determine the performance impact of distributing compute.

(2) In previous work, the under-utilization of compute resources resulting from offloading bandwidth-intensive parts of computations to in-memory co-processors is usually not a concern since the cumulative in-memory compute resources are very small compared to the main processing unit (6% in [15]) . However, a node in an exascale system will be connected to a large memory network [22] and cumulative in-memory compute resources can range from 10%-40% (1-4 SMs per stack) of the main processing unit. Considering this, we view all compute resources integrated with memory as a part of a multi-unit NUMA system rather than just co-processors. The work in [9] discusses under-utilization of PIM units, but chooses to mitigate the problem by executing concurrent kernels in in-memory compute units provided that there is no dependency among kernels. By contrast, we launch a single



(a) Lulesh: IntegrateStressForElems.



(b) Lulesh: CalcLagrangeElementsPart2.

Figure 10: Lulesh kernels memory access patterns.

kernel across the node, distributing the threadblocks among different SMs without any restrictions on dependencies.

Having in-package high-bandwidth memory with compute units results in a NUMA-like system. Previous works have investigated how an application (both its threads [16] and data [17, 18, 28]) should be mapped onto a NUMA system. These works assume a homogeneous system where each unit is identical and compute is spread across all the units. By contrast, our work aims at finding suitable architectures (homogeneous, heterogeneous or centralized) for a given application.

Researchers have previously investigated ways to better exploit the heterogeneity in memory resulting from multiple memory levels. These works typically assume a centralized compute and propose how the data should be mapped across different levels of memory to obtain better performance. There have been proposals ranging from using in-package 3D stack memory as another level of caches [29] to others advancing OS-supported [19] and OS-transparent [4, 14, 31] page mapping schemes for in-package memory. Our work instead aims to explore if centralized compute should be the architecture of choice in the first place, given that other architectures are now feasible.

6 CONCLUSIONS AND FUTURE WORK

Modern packaging technology has made in-package high-bandwidth 3D stack memory a reality. This has opened up opportunities for non-uniform processing-in-memory (NUPIM) architectures within exascale nodes. We study memory behavior of benchmarks and evaluate the performance of different architectures. Our experiments show that:

(1) No single architecture is better across all benchmarks. However, non-uniform distributed processing-in-memory architectures perform better in common cases.
(2) Distributed in-memory processing may provide performance benefits even in cases where it results in higher off-chip accesses.
(3) Benefits of non-uniform in-memory processing depend on various factors. Individual kernels of a single application can have varying speedup across different configurations, which can potentially be exploited to reduce power consumption.

We observe that having distributed in-memory compute might provide higher performance but also results in higher remote accesses (and power consumption). Furthermore, the engineering cost incurred in designing and verifying non-uniform in-memory compute is significant. In the future, we plan to integrate such overheads in our cost analysis and investigate ways to mitigate the increase in power consumption.

REFERENCES

[1] 2009. ATI and Nvidia face off–obliquely. (Oct. 2009). https://www.cnet.com/news/ati-and-nvidia-face-off-obliquely
[2] 2009. *GPGPUSim v3.2.2. GTX 480 Configuration.*
[3] E. Azharkish. 2015. High Performance AXI-4.0 Based Interconnect for Extensible Smart Memory Cubes. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE.
[4] C. C. Chou. 2014. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In *International Symposium on Microarchitecture (MICRO)*. IEEE, Cambridge, UK.
[5] Hybrid Memory Cube Consortium. 2015. *Hybrid Memory Cube Specification 2.1.* Technical Report.
[6] A. Bakhoda et al. 2009. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Boston, MA.
[7] A. Jog et al. 2013. OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance. In *International Conference on Architectural Support for Programming languages and Operating systems (ASPLOS)*. ACM, Houston, TX.
[8] A. Mariano et al. 2016. Analyzing and Improving Memory Access Patterns of Large Irregular Applications on NUMA Machines. In *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, Heraklion, Greece.
[9] A. Pattnaik et al. 2016. Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities. In *International Conference on Parallel Architectures and Compilation (PACT)*. IEEE, Haifa, Israel.
[10] B. Hong et al. 2016. Accelerating linked-list traversal through near-data processing. In *International Conference on Parallel Architectures and Compilation (PACT)*. IEEE, Haifa, Israel.
[11] I. Karlin et al. 2012. *LULESH Programming Model and Performance Ports Overview.* Technical Report. Lawrence Livermore National Lab.
[12] J. Ahn et al. 2015. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *International Symposium on Computer Architecture (ISCA)*. IEEE, Portland, OR.
[13] J. A. Stratton et al. 2012. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing.* Technical Report. IMPACT.
[14] J. H. Ryoo et al. 2017. SILC-FM: Subblocked InterLeaved Cache-Like Flat Memory Organization. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Austin, TX.
[15] K. Hsieh et al. 2016. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *International Symposium on Computer Architecture (ISCA)*. IEEE, Seoul, South Korea.
[16] M. Diener et al. 2013. Communication-based mapping using shared pages. In *International Symposium on Parallel Distributed Processing (IPDPS)*. IEEE, Boston, MA.
[17] M. Dashti et al. 2013. Traffic management: a holistic approach to memory placement on NUMA systems. In *International Conference on Architectural Support for Programming languages and Operating systems (ASPLOS)*. ACM, Houston, TX.
[18] M. Diener et al. 2015. Locality vs. Balance: Exploring Data Mapping Policies on NUMA Systems. In *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, Turku, Finland.
[19] M. Meswani et al. 2015. Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Burlingame, CA.
[20] S. Ashby et al. 2010. *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee.* Technical Report. U.S. Dept of Energy.
[21] S. Che et al. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *International Symposium on Workload Characterization (IISWC)*. IEEE.
[22] T. Vijayaraghavan et al. 2017. Design and Analysis of an APU for Exascale Computing. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Austin, TX.
[23] X. Jian et al. 2017. Understanding and Optimizing Power Consumption in Memory Networks. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Austin, TX.
[24] Mingyu Gao and Christos Kozyrakis. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Barcelona, Spain.
[25] M. Gokhale. 1995. Processing in Memory: The Terasys Massively Parallel PIM Array. *IEEE Computer* 28, 4 (April 1995).
[26] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *Symposium on VLSI Technology (VLSIT)*. IEEE.
[27] P. M. Kogge. 1994. EXECUBE-A New Architecture for Scalable MPPs. In *International Conference on Parallel Processing (ICPP)*. IEEE.
[28] Collin McCurdy and Jeffrey Vetter. 2010. Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, White Plains, NY.
[29] Mark Oskin and Gabriel H. Loh. 2015. A Software-Managed Approach to Die-Stacked DRAM. In *International Conference on Parallel Architectures and Compilation (PACT)*. ACM, Haifa, Israel.
[30] D. Patterson. 1997. A Case for Intelligent RAM. *IEEE Micro* 17, 2 (March 1997).
[31] J. Sim. 2014. Transparent hardware management of stacked DRAM as part of memory. In *International Symposium on Microarchitecture (MICRO)*. IEEE, Cambridge, UK.
[32] H. S. Stone. 1970. A Logic in Memory. *IEEE Transaction on Computers* C-19, 1 (Jan. 1970).