Near LLC Versus Near Main Memory Processing

Hossein Bitalebi
KTH Royal Institute of Technology
hobita@kth.se

Vahid Geraeinejad
KTH Royal Institute of Technology
vahidg@kth.se

Masoumeh Ebrahimi
KTH Royal Institute of Technology
mebr@kth.se

ABSTRACT

Emerging advanced applications, such as deep learning and graph processing, with enormous processing demand and massive memory requests call for a comprehensive processing system or advanced solutions to address these requirements. Near data processing is one of the promising structures targeting this goal. However, most recent studies have focused on processing instructions near the main memory data banks while ignoring the benefits of processing instructions near other memory hierarchy levels such as LLC. In this study, we investigate the near LLC processing structures, and compare it to the near main memory processing alternative, specifically in graphics processing units. We analyze these two structures on various applications in terms of performance and power. Results show a clear benefit of near LLC processing over near main memory processing in a class of applications. Further, we suggest an architecture, which could benefit from both near main memory and near LLC processing structures, but requiring the applications to be characterized in advance or at run time.

KEYWORDS

Near data processing, Processing near main memory, Processing near LLC, GPU, Irregular memory accesses

ACM Reference Format:
Hossein Bitalebi, Vahid Geraeinejad, and Masoumeh Ebrahimi. 2022. Near LLC Versus Near Main Memory Processing. In *The 14th Workshop on General Purpose Processing Using GPU (GPGPU'22), April 3, 2022, Seoul, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3530390.3532726

1 INTRODUCTION

Graphics processing unit (GPU) is one of the most popular platforms for executing *compute-intensive* applications due to their massive degree of parallel processing [24]. In a conventional GPU structure, shown in Figure 1, each processing core, called streaming multiprocessor (SM) in NVIDIA terms, is equipped with its own private L1 cache. Last level cache (LLC) and main memory are shared among all SMs, and they are accessible through an interconnection network. An SM core first sends its data requests to the L1 cache. If data is not available in this level, the request is sent to the lower levels of memory hierarchy i.e. LLC and main memory. If the LLC cache could provide the data, it sends back the response directly to the corresponding SM, otherwise, the main memory is responsible for responding the current request.

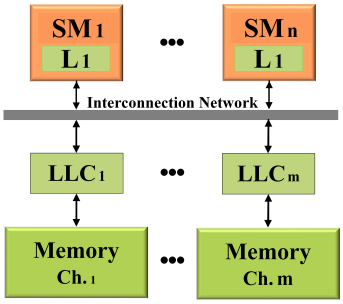

Figure 1: Conventional GPU architecture

When an application is assigned to a GPU, it is divided into various thread blocks (TBs) and each TB is assigned to an SM independently. Each TB consists of multiple independent warps, where each warp has several processing threads. The number of threads depends on the length of the GPU single-instruction-multiple-thread (SIMT) [21]. When an SM generates a memory request, its associated warp will be blocked until receiving the memory response. In the meantime, a ready warp continues its processes with the current SM. Now, if all assigned warps of an SM are blocked due to the late memory response time, the SM switches to the stall state until at least one warp is ready for execution. This is often a scenario in *memory-intensive* applications with a large number of memory requests. The situation can get worse if application memory requests follow an irregular memory access pattern. The term "irregular" refers to a circumstance in which the locality property of memory requests is very low and the variety of required data blocks is very high [5]. During the execution of such applications, memory requests are divergent, and consecutive instructions usually call different memory data blocks. This means the coalescing rate of such applications is low, and the cache controller should frequently evict an in-use data block and replace it with a new one.

To relax this issue, Near Data Processing (NDP) is introduced with the idea of placing some processing cores near the main memory, capable of executing the offloaded instructions from SMs. This structure could alleviate the stalling SMs challenge, thus speeding up the memory-intensive applications. NDP could not have been employed in traditional memory structures due to the lack of processing cores. With the advent of new memory technologies, such as 3D-stacked memory, processing cores can be integrated near the main memory; therefore, providing the opportunity to process the instructions near the data banks.

Most of the previous works [18, 26, 33, 34, 37] have focused on investigating different Near main Memory Processing (NMP) structures in GPU. However, few recent studies considered the benefits of processing instructions near the intermediate levels

GPGPU'22, April 3, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9348-5/22/04.
https://doi.org/10.1145/3530390.3532726

of the memory hierarchy [25, 27]. In line with these works, in this paper, we investigate near Last Level Cache (LLC) processing, hereafter refereed to as NLP, while it can be extended to other cache hierarchy levels. We characterize various applications and analyze their suitability for execution near main memory or LLC. The following are the major contributions of this paper:

- Various applications are analyzed, showing a high rate of L1 and LLC cache misses in GPU, and a clear difference between L1 and LLC miss rates in some applications.
- An NLP structure is explained and simulated to process the offloaded instructions near the LLC data banks. This structure is then compared with its NMP counterpart. Further a joint structure is suggested to combine the benefits of NLP and NMP structures.
- Experiments are conducted to show the benefits of NLP and NMP over conventional GPU in different classes of applications in terms of performance and power. The obtained results suggest an opportunity to characterize applications and offload them for processing either near the LLC or main memory to optimize their performance and power.

2 RELATED WORK

The efficiency of near data processing not only depends on whether offloaded instructions are selected properly but also if they are executed in the suitable memory hierarchy level. Otherwise, NDP may lead to inefficiency and impose extra overhead to the system. To this end, numerous attempts have been made to perform efficient NDP. The following are some of the studies which are reviewed.

Near main memory processing: As previous studies have investigated, advancements in memory technology provide the opportunity to process the instructions near main memory data banks [2, 4, 6, 13, 15, 16, 35]. Ahn et al. [2] proposes a hardware-based architecture without changing the programming paradigm to determine whether an instruction should be executed in memory or on processors, depending on the locality of data. Elliot et al. [22] proposes a programming interface that facilitates near data bank computation based on the locality property. Mahmut et al. [19] tries to take advantage of near data computing by different compiler schemes. Tang et al. [30] also provides compiler support that partitions the computations with the goal of reducing the distance-to-data on the on-chip network. Hsieh et al. [17] statically determines the instructions which should be offloaded into the auxiliary cores during the application execution. In order to relax the inter stack communication, Kim et al. [20] suggests a network-on-chip to allow data to be moved across different stacks.

Near cache processing: As mentioned earlier, most of the NDP efforts are implemented as near main memory processing. Few recent studies have focused on taking advantage of processing the instructions near intermediate levels of the memory hierarchy, such as LLC [11, 31]. Denzler et al. [11] proposes near cache accelerator that its compute units are directly connected to LLC. Vieira et al. [31] proposes a near cache bank processing mechanism to speed up the convolutional neural network algorithm. This paper offloads the execution instructions related to the convolutional and pooling layers. Pattnaik et al. [27] explores eligible instruction sets for offloading to the last level cache. They also add compute units near the LLC data banks to execute the offloaded instructions.

Processing using storage cells: There is another set of work, investigating in-memory processing with the focus on changing the structure of memory cells [1, 8, 10, 12, 14, 32]. Aga et al. [1] proposes a Compute Cache unit to enable in-place computation in the cache by using bit-line SRAM circuit technology. Bit Prudent In-Cache Acceleration [32] proposes an SRAM based in-cache architecture to accelerate CNN inference by leveraging network redundancy. Neural-Cache [12] focuses on re-purposing cache structures to transform them into compute units capable of fully executing convolutional, fully connected, and pooling layers in the inference mode. Our study, however, is mainly focused on processing instructions near the data banks without any change to the structure of memory cells. Nevertheless, it is extensible to in-memory processing structures which enable the instruction execution by interactions between cells.

3 A NEW LOOK INTO NEAR DATA PROCESSING IN GPU

GPU is one of the most efficient hardware platforms that play a significant role in speeding up compute and memory intensive applications [23]. Various studies have focused on the NDP structure to overcome the memory bandwidth bottleneck of the processing systems. However, NDP is still in its infancy and there is a lot of room to fully utilize this structure.

We start this section by investigating the behaviour of various applications with regard to L1 and LLC cache miss rates in GPU. Then, we study the two main structures of near data processing including: Near LLC Processing (NLP) and Near main Memory Processing (NMP). More specifically, we describe the NLP processing and compare it to its more traditional counterpart NMP. Finally, we classify different applications based on their cache access pattern and investigate their suitability for execution near LLC or main memory.

3.1 Motivation

Many light processing cores of the conventional GPU make the concurrent processing of thousand of threads possible. Despite its advantages, cache structure in conventional GPU is vulnerable to irregular data accesses [9]. While the L1 cache miss rate in Central Processing Unit (CPU) is around 6.3% to 33% [28], this value is around 85% in GPU, based on our experiments depicted in Figure 2. We have performed a set of analysis on various applications (listed in Table 5), simulated on a conventional GPU (see Table 2 for more information).

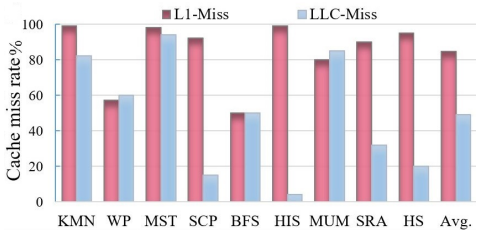

Figure 2: Conventional GPU cache miss access rate

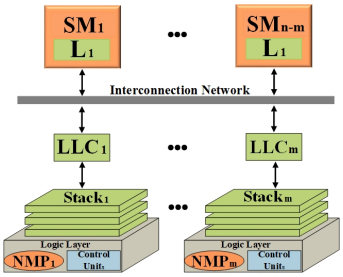

Figure 3: Near main memory processing structure

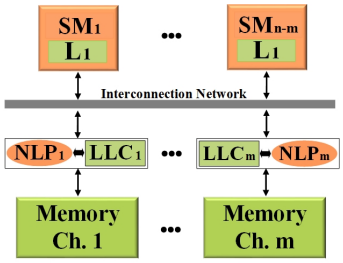

Figure 4: Near LLC processing structure

As shown in Figure 2, for the examined applications, L1 and LLC cache miss rates are very high, with an average of 85% and 50%, respectively. Another interesting observation is the relatively high L1 miss rate and low LLC miss rate in applications such as SCP, HIS, SRA, and HS. This observation motivated us to investigate the suitability of near LLC processing as data is likely resided in this level of the memory hierarchy. This is in contrast to near main memory processing in conventional NDP structures.

Lastly, due to the high cache miss rates in GPU, we have seen a better potential to benefit from NDP than in CPU, so we have chosen GPU as our main target platform for implementing NLP.

3.2 NMP vs. NLP Processing

In the Near Data Processing (NDP) structure, SM cores may offload some instructions to other memory hierarchy levels for execution. For this purpose, some auxiliary processing cores should be placed near the memory data banks to process the offloaded instructions. As a general categorization, NDP can be divided into two main classes: Near main Memory Processing (NMP) and Near LLC Processing (NLP). Between the two, NMP has been extensively studied in literature by adding cores in the logic layer of the emerging 3D memory structures.

Figure 3 shows a conventional NMP structure. Logic layers of the 3D-stacked memory are equipped with processing cores to execute the offloaded instructions. The instructions that should be offloaded are usually detected and labeled during the compilation time. In SM cores, after an instruction is fetched, the decoder unit checks whether the instruction is labeled as "to be offloaded" (into the main memory cores) or not. Most NDP studies are focused on the NMP structure without considering the advantages of processing instructions near intermediate levels of the memory hierarchy.

Figure 4 demonstrates an NLP structure. As opposed to NMP, main memory data banks are not considered as targets of offloaded instructions. Instead, auxiliary cores are located near the LLC data banks, and offloaded instructions use the settled data within the LLC blocks, directly. As shown in Figure 2, applications with regular LLC requests (such as SCP, HIS, SRA, and HS) are more likely to gain proficiency by NLP execution. Regular LLC requests lead to low LLC miss access and most of the L1 missed requests are responded by the LLC during the application execution. This means most of the data demand by offloaded instructions are provided by the LLC.

It is worth noting that although both of the NMP and NLP structures are equipped with auxiliary cores to process the offloaded instructions, the offloaded instructions at NLP pass fewer steps to be processed near data banks compared to its NMP counterpart.

3.3 Characterizing Applications

As can be seen in Figure 2, cache access shows various patterns among different applications and different cache levels. We have classified and listed applications based on these patterns in Table 1. Based on this classification, applications in Class 1 and 2 have low or high cache miss access for both the L1 and LLC (such as KMN, MST, and MUM) due to their inherent compute sensitivity and data access irregularity rate. Applications in Class 3 and 4 have high L1 miss and low LLC miss access and vice versa (such as SCP, HIS, SRA, and HS). There is another class of applications, Class 5, showing a moderate amount of miss access for both L1 or LLC (such as WP and BFS).

Table 1: Cache access pattern classification

Miss Access rate	L1	LLC
Class 1	Low	Low
Class 2	**High**	**High**
Class 3	**High**	**Low**
Class 4	Low	High
Class 5	Moderate	Moderate

Applications categorized in Class 1 and 4 of Table 1 are usually more suitable for processing in SM cores similar to that of the conventional GPU due to the fact that the L1 miss rate is low, and data is likely to be hit there. In other words, these applications may not benefit from processing near data banks, neither NLP nor NMP.

Applications that belong to Class 2 are more suitable for near main memory processing as high LLC miss rate means a high number of accesses to the main memory. Thereby, it is more advantageous for instructions to be offloaded to the cores near the main memory, thus motivating the NMP choice.

The applications in Class 3 have a stronger potential for executing offloaded instructions near LLC due to low LLC miss rate. For such applications, NLP is a better choice than NMP by eliminating the long service time of the main memory and shortening the data path.

In the case of Class 5 applications, there is no clear benefit from processing on SMs or cores near other memory levels. As it will be discussed in Section 5, evaluation, these applications could still benefit from NDP processing, either NMP or NLP. This is mainly because of the overall high miss rates in GPU.

Table 2: Baseline GPU configuration

Parameter	Value
Total cores	56 Streaming Multiprocessors (SM)
Per core	48 warps, 32 warp width, 8 CTAs, 1536 threads, 32768 registers, 48 KB scratchpad memory
L1 data cache	32 KB, 4-way, 128B block size
LLC (L2) cache	8 x 128K, 16-way
Memory	FR-FCFS scheduler, DDR3-1333H, 8 memory channel
Core, L2 clock	700 MHz, 700 MHz
Interconnect	2D mesh, XY routing, 1 core/node, 4 VCs, 4 routing latency, 1 channel latency

Table 3: NMP specific configuration

Parameter	Value
Total cores	56
Main cores	48 SMs
Near main memory cores	8
Memory stack Configuration	8 memory stacks, 16 vaults/stack, 16 banks/vault, 64 TSVs/vault
Intra-stack BW	160 GB/s per stack
Inter-stack BW	40 GB/s per link, fully connected
GPU to memory BW	80 GB/s per link

Table 4: NLP specific configuration

Parameter	Value
Total cores	56
Main cores	48 SMs
Near LLC cores	8

4 METHODOLOGY

In order to have a comprehensive evaluation, both NLP and NMP structures are implemented by GPGPU-Sim v3.2.2 [3], a cycle accurate simulator. We have modified this simulator in line with our targets in this paper. In addition, we have used various benchmarks from different benchmark suits as Rodinia [7], Parboil [29], and CUDA SDK [36]. Table 2 demonstrates the configuration of the baseline GPU structure while additional configurations specific to NMP and NLP structures are shown in Table 3 and Table 4, respectively. To ensure fair comparison, the total number of processing cores, the size of cache memory, and the size of main memory are kept similar in each of the three configurations, including conventional GPU, NLP, and NMP. For example in the experiments, 56 SM cores are allocated in the Baseline GPU configuration (Table 2) whereas in the NMP (Table 3) and NLP (Table 4) structures, the number of SM cores is reduced to 48, and instead 8 cores are placed either near main memory or LLC. This keeps the total number of cores equal across all configurations.

As described in Table 2, in all mentioned configurations, each main core has a 32KB private L1 data cache and has access to 8*128KB shared L2 cache through the interconnection network.

Table 5: Benchmarks specifications

Abrr. (type)	Name	Suite	Description
KMN (Class 2)	Kmeans	Rodinia	k-means, clustering
WP (Class 5)	Weather prediction	CUDA	Numerical weather prediction
MST (Class 2)	Mergesort	CUDA	Parallel merge sort
SCP (Class 3)	Scalar product	CUDA	Scalar product calculations
BFS (Class 5)	Breadth-first search	Rodinia	Breadth-first search on a graph
HIS (Class 3)	Histogram	Parboil	Histograms for analysis
MUM (Class 2)	Mummergpu	Rodinia	Pairwise local sequence alignment
SRA (Class 3)	SRAD	Rodinia	Speckle reducing anisotropic diffusion
HS (Class 3)	Hotspot	Parboil	Processor temperature estimation

The Baseline architecture and NLP are adjusted to conventional DRAM technology and the bandwidth between the GPU main cores and main memory is set to 80 GB/s per link similar to the NMP architecture. On the other hand, NMP uses 3D-stacked memory technology with Intra-stack and Inter-stack per-link bandwidths of 160 GB/s and 40 GB/s, respectively.

5 EVALUATION

This study is mainly focused on different structures of NDP where some instructions are offloaded to lower memory hierarchy levels. In this section, the execution behavior of several applications is evaluated in terms of performance and power to determine which hierarchy level (near LLC data banks or near main memory data banks) is more desirable for processing the offloaded instructions.

5.1 Performance

Figure 5 compares the performance of the Baseline GPU configuration with those of NLP and NMP. Light and dark blue bars are the performance representative of NLP and NMP, respectively, normalized to that of the Baseline. The selected applications cover Class 2, 3, and 5 in Table 1 that could benefit from NDP, which includes both high and moderate rate of the L1 miss pattern. Applications in Class 1 and 4 have a low L1 miss rate, so the main cores are the best place to process them, and near data processing is of no interest. For simplicity, we excluded these two classes from our experiments.

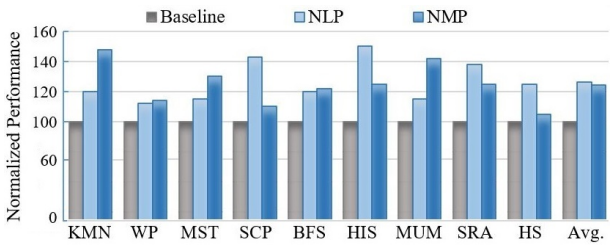

Figure 5: Performance comparison between Baseline, NLP and NMP

In Class 2 and 3, applications generate a significant amount of memory requests with poor locality through the L1 cache, and consequently they have a high rate of L1 miss access during the instruction execution. In addition to the high L1 miss rate, applications of Class 2 (such as KMN, MST and MUM) have a high LLC miss rate as well. As explained in Section 3, these applications could benefit more from near main memory processing as their required data does not exist in the cache lines, so ultimately the main memory has to respond to them. In this case, NLP is not an ideal choice and may impose extra delay by complicating the memory access path. Results from the Class 2 applications show that NMP outperforms the Baseline and NLP alternatives by 40% and 23%, respectively.

In contrary, applications of Class 3 (such as SCP, HIS, SRA, and HS) enjoy a low LLC miss rate. This means that the required data for the offloaded instructions are likely available in this cache level, and thereby, cores near LLC could benefit from the straightforward access to the cache lines. In this case, processing the offloaded instructions near the LLC data banks could eliminate the long service delay of the main memory. As a result, these applications are more likely to achieve better speed up in NLP. As can be seen in Figure 5, processing offloaded instructions by NLP in the applications of Class 3 leads to an average 40% and 24% higher performance than the Baseline and NMP alternatives, respectively. The performance improvement is partly due to bypassing the crowded main memory queues, long row switching procedure, and inter-stack communication overheads.

In applications of Class 5 (such as WP and BFS), however, both L1 and LLC miss rates are moderate. That means, nearly 50% of the L1 missed requests are responded by LLC and the other 50% will be responded by the main memory. As demonstrated in Figure 5, NLP and NMP structures show equal efficiency while both outperform the Baseline architecture, for instance by 20%, under the BFS benchmark.

5.2 Power Consumption

We have compared NLP and NMP in terms of the power consumption as well, shown in Figure 6. The results are reported based on the Instruction per Joule (Ins/J) and normalized to the Baseline architecture. As is clear, for the selected benchmarks, an NDP-equipped GPU (NLP or NMP) is more power efficient than the conventional GPU architecture. On average, NLP and NMP achieve better power efficiency by almost 11%.

The HIS benchmark, as a representative of Class 3 and suitable for NLP, is more power efficient than the Baseline and NMP by 15% and 6%, respectively. The benefits mainly rely on eliminating the long response time of the main memory including the queuing delay and service time.

The KMN benchmark, as a representative of Class 2 and more suitable for NMP, leads to a power efficiency of 18% and 13% compared to its Baseline and NLP counterparts. As it is expected, NLP may not lead to superior power efficiency in this benchmark. This owes to the insignificant role of LLC in providing the required date, and as a result, NLP cores should request the data from the main memory instead, which leads to higher power consumption.

The WP and BFS benchmarks in Class 5, also show a power efficiency of 9%, on average, with NLP or NMP over the Baseline.

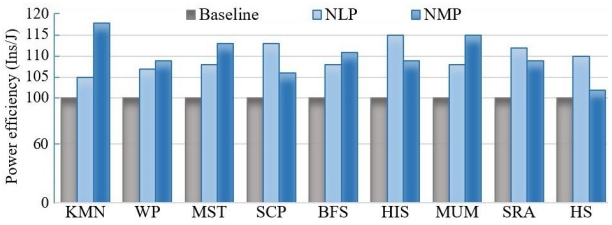

Figure 6: Power comparison between Baseline, NLP and NMP

5.3 Discussion

After an in-depth analysis of our results, we came to the following secondary reasons affecting the performance and power consumption of both NLP and NMP: 1) Offloading instructions to NLP or NMP leaves more space for data in the L1 cache, improving locality principle. This, in turn, reduces the L1 miss access rates and the costly cache victim rate; 2) In addition to the inherent ability of NLP and NMP in reducing the data movement, the involved overhead such as link traversal and the racing rate over the shared resources also reduces; and 3) Simplicity and proximity of data access for the offloaded instructions accelerate the application execution. Beside all the benefits, we should mention that NDP may impose extra area and process overhead to determine the instructions that are to be offloaded and also for offloading them into the auxiliary cores.

6 FUTURE PERSPECTIVE

In this paper, we have seen the benefits of NLP and NMP for different classes of applications. Consequently, we suggest that an optimal architecture should take advantages of both structures. Our suggested architecture is illustrated in Figure 7, equipped with auxiliary cores near both LLC and main memory. Using this architecture, instructions can be either offloaded to NLP or NMP directly. Thereby, the number of accesses to the main memory could be reduced when NLP is chosen, whereas intermediate cache levels could be bypassed when NMP is selected. It is expected that this joint architecture could optimize the overall performance and power to a great extent. However, for this to happen, either applications should be characterized offline or the decisions on which instructions to be offloaded and where to offload them (NLP or NMP) have to be made dynamically at run time. This is in contrast to the current static compilation time decision making process. Although offline application characterization is a viable approach, for a better generality and scalability, as a part of our future work, we will investigate dynamic instruction offloading for this joint architecture.

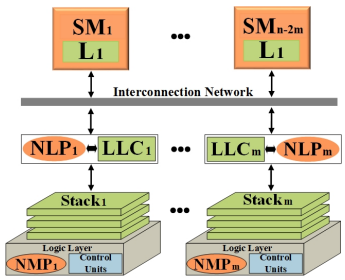

Figure 7: A joint architecture, combining NLP and NMP

7 CONCLUSION

While GPU exhibits high efficiency in executing various applications, they usually show high cache miss rates in memory intensive applications, when compared to CPU. Near data processing (NDP) is one of the solutions to alleviate this problem by processing instructions near storage units. Two main approaches of NDP are near LLC processing (NLP) and near main memory processing (NMP). In this paper, we characterized applications and claimed that NLP and NMP structures behave differently on different classes of applications. After analyzing applications in terms of performance and power, this claim has been confirmed, showing the benefits of both structures but on different classes of applications. As an optimal solution, we suggested a structure that offloads instructions directly to NLP or NMP. For this to happen, applications should be characterized offline. Dynamic instruction offloading was introduced as a scalable alternative, which will be investigated in our future work.

REFERENCES

[1] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. 2017. Compute caches. In *2017 IEEE International Symposium on HPCA*. IEEE, 481–492.

[2] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *2015 ACM/IEEE 42nd Annual International Symposium on ISCA*. IEEE, 336–348.

[3] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *2009 IEEE international symposium on ISPASS*. IEEE, 163–174.

[4] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, et al. 2018. Google workloads for consumer devices: Mitigating data movement bottlenecks. In *Proceedings of 23rd Conference on ASPLOS*. 316–331.

[5] Martin Burtscher, Rupesh Nasre, and Keshav Pingali. 2012. A quantitative study of irregular programs on GPUs. In *2012 IEEE International Symposium on IISWC*. IEEE, 141–151.

[6] Damla Senol Cali, Gurpreet S Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, et al. 2020. Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis. In *53rd IEEE/ACM Symposium on MICRO*. IEEE, 951–966.

[7] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on IISWC*. Ieee, 44–54.

[8] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 27–39.

[9] Kyu Hyun Choi and Seon Wook Kim. 2015. Study of cache performance on GPGPU. *IEIE Transactions on Smart Processing and Computing* 4, 2 (2015), 78–82.

[10] Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, and Jun Yang. 2018. Dracc: a dram based accelerator for accurate cnn inference. In *Proceedings of the 55th annual design automation conference*. 1–6.

[11] Alain Denzler, Rahul Bera, Nastaran Hajinazar, Gagandeep Singh, Geraldo F Oliveira, Juan Gómez-Luna, and Onur Mutlu. 2021. Casper: Accelerating Stencil Computation using Near-cache Processing. *arXiv preprint arXiv:2112.14216* (2021).

[12] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. 2018. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *2018 ACM/IEEE 45Th annual international symposium on ISCA*. IEEE, 383–396.

[13] Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2021. Syncron: Efficient synchronization support for near-data-processing architectures. In *2021 IEEE International Symposium on HPCA*. IEEE, 263–276.

[14] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. 2021. SIMDRAM: a framework for bit-serial SIMD processing using DRAM. In *Proc. of the 26th Conference on ASPLOS*. 329–345.

[15] Maryam S Hosseini, Masoumeh Ebrahimi, Pooria Yaghini, and Nader Bagherzadeh. 2021. Application Characterization for Near Memory Processing. In *2021 29th Euromicro International Conference on PDP*. IEEE, 148–152.

[16] Maryam S Hosseini, Masoumeh Ebrahimi, Pooria Yaghini, and Nader Bagherzadeh. 2021. Near Volatile and Non-Volatile Memory Processing in 3D Systems. *IEEE Transactions on Emerging Topics in Computing* (2021), 1–1. https://doi.org/10.1109/TETC.2021.3115495

[17] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W Keckler. 2016. Transparent offloading and mapping (TOM) enabling programmer-transparent near-data processing in GPU systems. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 204–216.

[18] Mohamed Assem Ibrahim, Hongyuan Liu, Onur Kayiran, and Adwait Jog. 2019. Analyzing and leveraging remote-core bandwidth for enhanced performance in GPUs. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 258–271.

[19] Mahmut Taylan Kandemir, Jihyun Ryoo, Xulong Tang, and Mustafa Karakoy. 2021. Compiler support for near data computing. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 90–104.

[20] Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, and Kevin Hsieh. 2017. Toward standardized near-data processing with unrestricted data placement for GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[21] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. 2008. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE micro* 28, 2 (2008), 39–55.

[22] Elliot Lockerman, Axel Feldmann, Mohammad Bakhshalipour, Alexandru Stanescu, Shashwat Gupta, Daniel Sanchez, and Nathan Beckmann. 2020. Livia: Data-centric computing throughout the memory hierarchy. In *Proceedings of the Twenty-Fifth Conference on ASPLOS*. 417–433.

[23] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2021. BatchSizer: Power-Performance Trade-off for DNN Inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC '21)*. 819–824.

[24] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2022. Coordinated Batching and DVFS for DNN Inference on GPU Accelerators. *IEEE Transactions on Parallel and Distributed Systems* 33, 10 (2022), 2496–2508. https://doi.org/10.1109/TPDS.2022.3144614

[25] Anant V Nori, Rahul Bera, Shankar Balachandran, Joydeep Rakshit, Om J Omer, Avishaii Abuhatzera, Belliappa Kuttanna, and Sreenivas Subramoney. 2021. REDUCT: Keep it Close, Keep it Cool!: Efficient Scaling of DNN Inference on Multi-core CPUs with Near-Cache Compute. In *2021 ACM/IEEE 48th Annual International Symposium on ISCA*. IEEE, 167–180.

[26] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K Mishra, Mahmut T Kandemir, Onur Mutlu, and Chita R Das. 2016. Scheduling techniques for GPU architectures with processing-in-memory capabilities. In *Proceedings of International Conference on Parallel Architectures and Compilation*. 31–44.

[27] Ashutosh Pattnaik, Xulong Tang, Onur Kayiran, Adwait Jog, Asit Mishra, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. 2019. Opportunistic computing in gpu architectures. In *2019 ACM/IEEE 46th Annual International Symposium on ISCA*. IEEE, 210–223.

[28] Aashish Phansalkar, Ajay Joshi, and Lizy K John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *Proceedings of the 34th annual international symposium on Computer architecture*. 412–423.

[29] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. 2012. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing* 127 (2012), 27.

[30] Xulong Tang, Orhan Kislal, Mahmut Kandemir, and Mustafa Karakoy. 2017. Data movement aware computation partitioning. In *Proceedings of the 50th Annual IEEE/ACM Symposium on MICRO*. 730–744.

[31] Joao Vieira, Nuno Roma, Gabriel Falcao, and Pedro Tomás. 2020. Processing convolutional neural networks on cache. In *ICASSP 2020-2020 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 1658–1662.

[32] Xiaowei Wang, Jiecao Yu, Charles Augustine, Ravi Iyer, and Reetuparna Das. 2019. Bit prudent in-cache acceleration of deep convolutional neural networks. In *2019 IEEE International Symposium on HPCA*. IEEE, 81–93.

[33] Yudong Wu, Mingyao Shen, Yi-Hui Chen, and Yuanyuan Zhou. 2020. Tuning applications for efficient GPU offloading to in-memory processing. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–12.

[34] Amir Yazdanbakhsh, Choungki Song, Jacob Sacks, Pejman Lotfi-Kamran, Hadi Esmaeilzadeh, and Nam Sung Kim. 2018. In-dram near-data approximate acceleration for gpus. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. 1–14.

[35] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. 2020. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits* 55, 6 (2020), 1733–1743.

[36] Cyril Zeller. 2011. Cuda c/c++ basics. *NVIDIA Coporation* (2011).

[37] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: Throughput-oriented programmable processing in memory. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. 85–98.