Optimizing GPU Cache Policies for MI Workloads*

Johnathan Alsop*, Matthew D. Sinclair*†, Srikant Bharadwaj*, Alexandru Dutu*, Anthony Gutierrez*, Onur Kayiran*, Michael LeBeane*, Brandon Potter*, Sooraj Puthoor*†, Xianwei Zhang*, Tsung Tai Yeh‡, Bradford M. Beckmann*

*AMD Research, †University of Wisconsin – Madison, ‡Purdue University

***Abstract*—In recent years, machine intelligence (MI) applications have emerged as a major driver for the computing industry. Optimizing these workloads is important, but complicated. As memory demands grow and data movement overheads increasingly limit performance, determining the best GPU caching policy to use for a diverse range of MI workloads represents one important challenge. To study this, we evaluate 17 MI applications and characterize their behavior using a range of GPU caching strategies. In our evaluations, we find that the choice of caching policy in GPU caches involves multiple performance trade-offs and interactions, and there is no one-size-fits-all GPU caching policy for MI workloads. Based on detailed simulation results, we motivate and evaluate a set of cache optimizations that consistently match the performance of the best static GPU caching policies.**

Keywords—execution-driven simulation, GPU caching, machine intelligence, machine learning

I. Introduction

In recent years, MI has emerged as an important driver for the computing industry. This has motivated a large amount of work optimizing hardware for MI, especially for Convolutional Neural Networks (CNNs) (e.g., [13]-[27]). Although these works have led to significant improvements in performance and energy efficiency of CNNs on modern multi-core CPUs, GPUs, and accelerators, it is challenging to analyze how future architectures will perform for these workloads. Here we focus on GPUs, as they are widely used for running MI workloads in numerous domains.

Although many MI systems use large discrete non-coherent GPUs, the emerging trend is to unify the CPU-GPU memory system [37]. Such a system is easier to program and can significantly reduce unnecessary data movement when GPU kernel launches are frequent, as can be the case with many MI workloads. However, implementing efficient coherent caches between CPUs and GPUs remains a significant challenge. GPUs use a coherence strategy which prioritizes memory throughput and scalability, sometimes at the cost of cache reuse. In an effort to better understand the trade-offs of different caching strategies, we evaluate the performance of these applications with multiple levels of GPU caching enabled using the publicly available AMD gem5 APU simulator [3].

We find that there is no best performing caching policy for all MI workloads. Although caching can significantly improve performance by enabling local data reuse, in some cases it can lead to harmful cache stalls and DRAM row locality disruption. Motivated by these results, we model and evaluate three microarchitectural optimizations which work together to mitigate the caching inefficiencies encountered by MI workloads. The first optimization avoids blocking for cache allocation, which reduces cache stalls. The second optimization applies a state-of-the-art CPU cache rinsing technique [41] to the last-level GPU cache to improve row buffer locality. Finally, we use a PC-based bypass prediction technique [40] to address remaining caching overheads while still caching accesses that can benefit from reuse. Together, these optimizations achieve the benefits of GPU cache reuse while minimizing caching overheads for these important MI workloads.

II. MI Background

Although there are many different MI methods, in this work we focus on deep neural networks (DNNs), which are some of the most commonly used MI workloads and are well-supported by the MIOpen library [29]. In state-of-the-art MI kernels, the tiling pattern, work item/work group parallelism, and scratchpad memory usage can vary even for a given layer based on what the framework determines is optimal for the target platform, making it difficult to generalize about the specific memory demands of any class of MI tasks. However, we summarize the high-level memory properties of the layers studied below; a more detailed discussion can be found in an extended version of this work [1].

- **Activation**: Simple elementwise operations with low reuse.
- **Fully connected**: High reuse between distant elements, compute intensive.
- **Convolutional**: Fewer parameters and less computationally intensive than fully connected, but has reuse between adjacent elements. These layers generally dominate DNN execution time.
- **Pooling**: Low reuse (depending on stride), unbalanced read and write counts.
- **Normalization**: High reuse (depending on normalization dimension).
- **RNN**: Similar to fully connected, but weights are common among layers, leading to more potential reuse.

978-1-7281-4045-2/19/$31.00 ©2019 IEEE

*This paper was accepted as a short paper

III. CPU-GPU Cache Coherence Background

In this work, we explore the costs and benefits of caching in GPU MI workloads by simulating three caching policies that differ in how loads and stores are handled in GPU caches:

- Uncached: Loads and stores bypass all GPU caches.
- CacheR: Loads are cached in L1 and L2, but stores bypass all GPU caches.
- CacheRW: Loads are cached in L1 and L2, stores bypass L1 and are combined in L2.

When load caching is disabled, read requests to the same cache line may be coalesced while the original bypass request is pending, but on a response the data is forwarded without being inserted in the cache. When load caching is enabled (CacheR, CacheRW), the GPU L1 and L2 caches always self-invalidate valid data at synchronization points (e.g., kernel boundaries) [47][49]. When store caching is enabled (CacheRW), stores still bypass the L1 but they may be delayed and coalesced at the L2 until a flush of all L2 dirty data is triggered at a system-scope synchronization point, at which time they are written back to memory [48].

IV. Methodology

A. The gem5 Simulator

To analyze MI workloads, we leverage the gem5 simulator [2][3]. In this work we focus on simulating open-sourced MI applications that use the MIOpen library [29] on top of the ROCm stack, including workloads from DNNMark [5], DeepBench [6][7], and MIOpen-benchmark [8]. Details of required application and simulator changes to run our experiments can be found in an extended version of this work [1].

B. System Configuration

Table 1 lists the key system parameters we simulate in gem5. Figure 1 shows the conceptual system design, which includes a 64-CU GPU with two levels of cache [30]. Our simulated GPU CU pipeline is based on AMD's GCN architecture [38] and uses the GCN3 ISA [31].

C. Applications

Table 2 shows the seventeen MI benchmarks that we studied. These MI benchmarks come from several popular MI suites: DNNMark [5], DeepBench [6][7], and MIOpen-benchmark [8]. We selected these benchmarks because they cover many different types of CNN and RNN layers and full NNs. A more detailed discussion of these applications can be found in an extended version of this work [1].

V. Caching Characterization

Figure 2 shows the execution time of each caching policy described in Section III for all applications, normalized to Uncached. Figure 3 shows the number of memory accesses that reach the DRAM controller, also normalized to Uncached. Overall, our results show that the best performing caching policy varies widely depending on the available

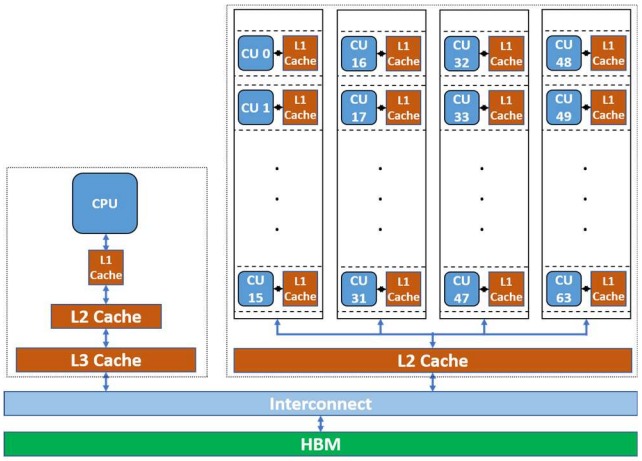

Figure 1: Overall simulated system.

Table 1: Key simulated system parameters.

GPU Parameters	
GPU Clock	1600 MHz
# of CUs	64
# SIMD units per CU	4
Max # Wavefronts per SIMD unit	10
VRF/SRF per SIMD unit	512/1600
CPU Parameters	
CPU Clock	4000 MHz
# CPUs	2
Memory Hierarchy	
GPU L1 D-cache per CU	16 KB, 64B line, 16-way write-through
GPU L1 I-cache per 2 CUs	32 KB, 64B line, 16-way
GPU L2 cache per 64 CUs	4 MB, 64B line, 16-way write-through (write-back for R data)
Main Memory	HBM2, 16 GB, 16 channels, 16 banks/channel, 1000 MHz, 512GB/s

Table 2: Studied MI workloads.

Application	Input	GPU Footprint
Backward Activation (BwAct) [5]	Batch size 100	2.4 GB
Backward Batch Normalization (BwBN) [5]	Batch size 512	5.88 MB
Backward Pool (BwPool) [5]	Batch size 256	252 MB
Backward Softmax (BwSoft) [5]	Batch size 512	0.02 MB
Composed Model (CM) [5]	Batch size 64	12.1 MB
Forward Activation (FwAct) [5]	Batch size 100	1.6 GB
Forward Batch Normalization (FwBN) [5]	Batch size 256	42 MB
Forward Fully Connected (FwFC) [5]	Batch size 512	148.2 MB
Forward LRN (FwLRN) [5]	Batch size 100	2.4 GB
Forward Pool (FwPool) [5]	Batch size 256	480 MB
Forward Softmax (FwSoft) [5]	Batch size 512	0.01 MB
SGEMM [6][7]	4Kx128x4K	68 MB
DGEMM [6][7]	4Kx128x4K	132MB
RNN Forward (FwLSTM/GRU) [6][7][8]	Batch size 1, sequence length 16, hidden layer 128, LSTM/GRU	0.38 MB
RNN Forward Backward (FwBwLSTM/GRU) [6][7] [8]	Batch size 1, sequence length 16, hidden layer 128, LSTM/GRU	0.48 MB

cache reuse and memory sensitivity of the workload. Workloads are grouped into three categories based on how they are affected by caching:

1. **Memory Insensitive (IN)**: Cache policy does not significantly affect overall execution time (<5% change) for CM, SGEMM, and DGEMM because the workload is compute bound or the potential for reuse is low.
2. **Reuse Sensitive (RS)**: Enabling caching consistently improves cache reuse and performance for FwBN, FwPool, FwSoft, BwSoft, BwPool, FwGRU, FwLSTM, FwBwGRU, FwBwLSTM, BwBN, and FwFC.
3. **Throughput Sensitive (TS)**: Enabling caching consistently hurts performance for FwAct, FwLRN, and BwAct due to a lack of cache reuse and high throughput demand for data.

A. *Caching Benefits: Reuse*

The main benefit of caching is that it enables cache reuse and therefore lower latency and higher bandwidth access to data. Layers with limited connectivity such as the pooling, convolution, and some normalization layers show limited benefit mainly because reuse is primarily between nearby work items and can be exploited even when caching is disabled. However, for workloads with higher connectivity, where reuse is possible between distant work items (e.g., FwFC, FwBN, FwBwGRU, and FwBwLSTM), we find that read caching can reduce memory demand by up to 93%. When the accesses that experience reuse are critical to performance (RS workloads), this reuse can also reduce execution time by up to 29%. In addition, write caching can further reduce memory demand by up to 71% and execution time by up to 32% for RS workloads which exhibit high potential for write coalescing at L2 such as BwPool and BwBN.

B. *Caching Overheads*

Although caching improves performance in many cases, for TS workloads it can increase execution time by up to 24%. We find that caching overheads manifest themselves primarily as 1) cache stalls due to added contention for cache resources, and 2) reduced DRAM row locality for requests that have been delayed in the caches. Although not shown here for space, an extended version of this paper includes detailed data on these overheads [1].

1) Cache Stalls: Cached requests require allocation on a miss, and this may cause stalls if all lines in the set are in a busy state (e.g., waiting for a pending load). In addition, coherence operations can add contention for shared resources such as tag and data arrays (e.g., due to failed cache allocation). High cache stall counts lead to worse execution time for FwAct and BwAct when read caching is enabled. FwPool also experiences high cache stalls, although negative

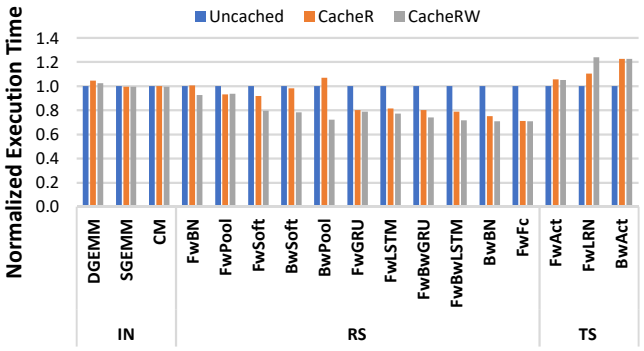

Figure 2: Execution time normalized to Uncached.

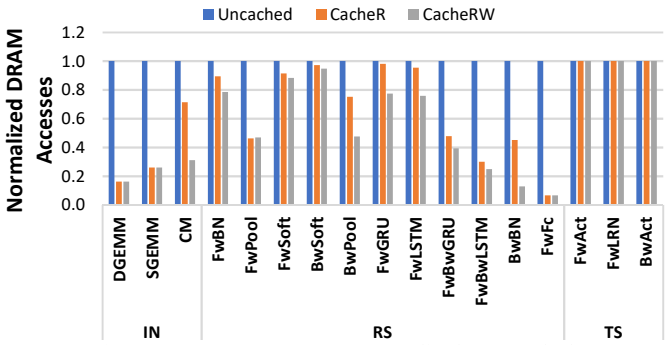

Figure 3: DRAM access counts normalized to Uncached.

effects here are offset by the added reuse achieved through caching.

2) DRAM Row Locality: Enabling read or write caching adds variability to memory access times through cache stalls described in Section V.B.1) or by delaying stores at the L2 so they can be coalesced (CacheRW). MI applications tend to have regular access patterns, and as a result enabling caching can interfere with this regularity and hurt DRAM row hit rates. In particular, FwPool, FwAct, FwLRN, and BwAct suffer from this effect. Although this effect in FwPool is outweighed by the benefits of cache reuse, for the TS workloads it contributes to a net performance degradation for caching configurations.

VI. Caching Optimizations for MI Applications

Motivated by the caching overheads we observe in GPU MI workloads, we next describe three potential architectural optimizations and evaluate their effect on performance. All are applied to the most aggressive caching policy, CacheRW, and are compared against the best and worst performing static configurations as measured in Figure 2. Figure 4 reports the normalized execution time for these optimizations.

A. *Allocation Bypass*

We begin by adapting our caching policies to converting cached requests to bypass requests whenever allocation would require blocking. This allocation bypass optimization is plotted as CacheRW-AB in *Figure 4*.

Non-blocking caching optimization reduces cache stalls per request significantly, but it has a minimal effect on overall performance for most applications. This can be explained by

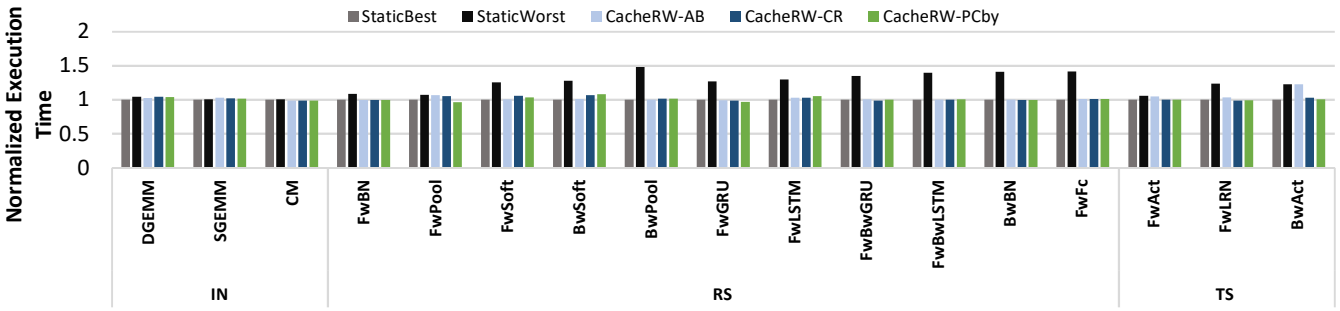

Figure 4: Execution time of best and worst static cache policy (from Figure 2) compared with allocation bypassing (CacheRW-AB), cache rinsing (CacheRW-CR), and PC-based bypassing (CacheRW-PCby) optimizations added. Normalized to best static configuration.

the fact that allocation bypassing does little to reduce the added congestion overhead, and in some cases adds to it by eliminating a throttling effect from the L1 cache level (this increases execution time by 7% for BwPool). The main exception is FwLRN, which sees significant benefits due to improved DRAM row hit ratios.

B. *Row Locality-Aware Cache Rinsing*

Although allocation bypass avoids row locality disruption caused by blocking allocation operations, it does not avoid disruption due to L2 write coalescing. To address this, we next add a row locality-aware cache rinsing scheme based on a method originally proposed for CPUs by Seshadri *et al.* [6]. This technique adds a dirty block index to the GPU L2 that tracks dirty blocks in each DRAM row. Whenever a dirty block is evicted, a writeback of all other dirty blocks in that row is triggered.

We add this cache rinsing optimization on top of the allocation bypassing optimization, denoted CacheRW-CR in Figure 4. Cache rinsing counteracts caching's detrimental effect on DRAM row locality overhead for affected (mainly TS) workloads, offering DRAM row hit rates that are even higher than those of the best static configuration. For example, as Figure 4 shows, cache rinsing reduces the performance overheads of caching for BwAct and FwAct.

C. *PC-Based L2 Bypassing*

We next attempt to address remaining performance overheads due to caching by predicting whether caching will be beneficial (i.e., whether cache reuse is likely), then dynamically choosing to use cached requests and incur the resulting overheads only when that is the case. Past work has explored this concept for adaptive load bypassing at the L1, proposing a PC-based reuse predictor to avoid cache pollution and more effectively use limited cache space [40]; we apply the same PC-based technique instead to the L2 for both loads and stores for the purpose of avoiding congestion overheads when reuse is unlikely.

We apply PC-based L2 bypassing on top of the allocation bypassing and cache rinsing optimizations and denote it as CacheRW-PCby in Figure 4. Overall, it is effective at predicting reuse for MI workloads. For nearly all workloads, the combination of allocation bypassing, cache rinsing, and PC-based bypassing matches or exceeds the performance of the best static cache configuration by selectively incurring cache overheads when they are expected to be beneficial.

VII. Related Work

There have been multiple prior efforts to enable efficient caching and coherence in GPUs [47][48][49][50][51][52][53]. In general, these aim to maximize cache reuse, often by avoiding unnecessary cache flush and invalidation actions. In contrast, this work analyzes cache overheads for workloads where reuse may be impossible.

Past work on adaptive cache bypassing [40][56][57], locality-aware rinsing [41][42][43][44], flexible coherence request types [54], and cross-layer coordination of scheduling and memory management [55] have also been shown to improve cache efficiency by matching caching policies to GPU workload demands. In contrast, the primary contribution of this work is to characterize the sources of cache inefficiency in GPU MI workloads and to describe techniques that address the specific caching overheads in this important domain.

Prior work has simulated MI workloads on in-house simulators [32][34][35][36], using analytical models [32][33], or on discrete GPU simulators [28][45][46]. In contrast, this work evaluates MI workloads on a cycle-level, publicly available simulator that does not suffer from the inaccuracies that arise with higher level ISAs [4].

VIII. Conclusions and Future Work

In this work we find that caching reads and writes has mixed behavior for MI workloads. For some workloads, it improves performance by up to 29% through increased cache reuse, but for others it degrades performance by up to 24% by incurring cache stalls and interfering with DRAM row locality. Based on the detailed performance overheads we identify by running open-source MI workloads on the AMD gem5 APU simulator, we implement and evaluate a set of adaptive GPU cache optimizations in gem5. These optimizations allow us to leverage the benefits of caching when it is helpful while avoiding performance overheads when caching hurts.

IX. Acknowledgements

The authors would like to thank Gabe Loh for his feedback and insights. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

REFERENCES

[1] Johnathan Alsop, Matthew D. Sinclair, Srikant Bharadwaj, Alexandru Dutu, Anthony Gutierrez, Michael LeBeane, Sooraj Puthoor, Xianwei Zhang, Tsung Tai Yeh, and Bradford M. Beckmann. Optimizing GPU Cache Policies for MI Workloads. https://arxiv.org/abs/1910.00134 [cs.AR]. September 2019.

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 Simulator. May 2011, In *ACM SIGARCH Computer Architecture News*.

[3] Anthony Gutierrez, Bradford M. Beckmann, Alexandru Dutu, Joseph Gross, John Kalamatianos, Onur Kayiran, Michael LeBeane, Matthew Poremba, Brandon Potter, Sooraj Puthoor, Matthew D. Sinclair, Mark Wyse, Jieming Yin, Xianwei Zhang, Akshay Jain, Timothy G. Rogers. Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level. In *Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, February 2018.

[4] Anthony Gutierrez, Bradford M. Beckmann, Sooraj Puthoor, Matthew D. Sinclair, Tuan Ta, and Xianwei Zhang. AMD gem5 APU simulator: Modeling GPUs Using the Machine ISA. *Tutorial at International Symposium on Computer Architecture*, June 2018.

[5] Shi Dong and David Kaeli. DNNMark: A Deep Neural Network Benchmark Suite for GPUs. In *Proceedings of the General Purpose GPUs* (GPGPU-10). 2017.

[6] Sharan Narang. DeepBench. https://svail.github.io/DeepBench/. September 2016.

[7] Sharan Narang and Greg Diamos. An update to DeepBench with a focus on deep learning inference. https://svail.github.io/DeepBench-update/. June 2017.

[8] Pat Flick. MIOpen-benchmarks. https://github.com/patflick/miopen-benchmark. October 2017.

[9] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. May 2015.

[10] AMD. HCC: An open source C++ compiler for heterogeneous devices. https://github.com/RadeonOpenCompute/hcc.

[11] Ben Sander, Greg Stoner, Siu-chi Chan, Wen-Heng Chung, Robin Maffeo. HCC: A C++ Compiler for Heterogenous Computing. HSA Foundation, Tech Report (2015).

[12] HIP: Heterogeneous-computing Interface for Portability. https://github.com/ROCm-Developer-Tools/HIP/.

[13] Amir Yazdanbakhsh, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks. *Proceedings of the 45th Annual Symposium on Computer Architecture (ISCA)*, June 2018.

[14] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, Gennady Pekhimenko. Gist: Efficient Data Encoding for Deep Neural Network Training. *Proceedings of the 45th Annual Symposium on Computer Architecture (ISCA)*, June 2018.

[15] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. A Configurable Cloud-Scale DNN Processor for Real-Time AI. *Proceedings of the 45th Annual Symposium on Computer Architecture (ISCA)*, June 2018.

[16] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[17] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, and Anand Raghunathan. 2017. ScaleDeep: A Scalable Compute Architecture for Learning and Evaluating Deep Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (ISCA '17). ACM, New York, NY, USA, pp. 13-26.

[18] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brueck Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks, *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, Toronto, ON, 2017, pp. 27-40.

[19] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN Accelerator Efficiency Through Resource Partitioning. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (*ISCA*). pp. 535-547

[20] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 548-560.

[21] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. 2017. Understanding and Optimizing Asynchronous Low-Precision Stochastic Gradient Descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (ISCA '17). ACM, New York, NY, USA, pp. 561-574.

[22] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: ineffectual-neuron-free deep neural network computing. In *Proceedings of the 43rd International Symposium on Computer Architecture* (ISCA '16). IEEE Press, Piscataway, NJ, USA, 1-13.

[23] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture* (ISCA '16). IEEE Press, Piscataway, NJ, USA, pp. 243-254.

[24] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John P. Strachan, miao Hu, R. Stanley Williams, and Vivek Srikumar. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars, *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016, pp. 14-26.

[25] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture* (ISCA '16). IEEE Press, Piscataway, NJ, USA, pp. 267-278.

[26] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyankiy, Liang Xiong, and Xiaodong Wang. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018, pp. 620-629.

[27] Yu-Hsin Chen, Joel Emer and Vivienne Sze, Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016, pp. 367-379.

[28] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2Sim: a simulation framework for CPU-GPU computing.

In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques* (*PACT*), pp. 335-34, 2012.

[29] Jehandad Khan, Paul Fultz, Artem Tamazov, Daniel Lowell, Chao Liu, Michael Melesse, Murali Nandhimandalam, Kamil Nasyrov, Ilya Perminov, Tejash Shah, Vasilii Filippov, Jing Zhang, Jing Zhou, Bragadeesh Natarajan, and Mayank Daga. MIOpen: An open source library for deep learning primitives. https://arxiv.org/abs/1910.00078, September 2019.

[30] AMD Radeon Technology Group. Radeon's next-generation Vega architecture. https://radeon.com/_downloads/vega-whitepaper-11.6.17.pdf, November 2017.

[31] AMD. Graphics Core Next Arcitecture, Generation 3. https://gpuopen.com/compute-product/amd-gcn3-isa-architecture-manual/. August, 2016.

[32] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeffrey Pool, Youngeun Kwon and Stephen W. Keckler. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. In *2018 IEEE International Symposium on High Performance Computer Architecture* (*HPCA*), Vienna, 2018, pp. 78-91.

[33] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Bruckey Khailany, Joel Emer, and Stephen Keckler. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture* (*ISCA*), Toronto, ON, 2017, pp. 27-40.

[34] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz and William J. Dally. EIE: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA),* Seoul, 2016, pp. 243-254

[35] Jorge Albericio, Patrick Judd, Alberto Delmas, Sayeh Sharify and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture,* Boston, 2017, pp. 382-394.

[36] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting Numerical Precision Variability in Deep Neural Networks. In *Proceedings of the 2016 International Conference on Supercomputing* (ICS '16).

[37] Luke Durant, Olivier Giroux, Mark Harris, and Nick Stam. Inside Volta: The World's Most Advance Data Center GPU. https://devblogs.nvidia.com/inside-volta/?ncid=so-lin-vt-13919, May 2017.

[38] AMD. AMD Graphics Core Next (GCN) Architecture. https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf , June 2012.

[39] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. DeepCPU: Serving RNN-based Deep Learning Models 10x Faster. In *Proceedings of the 2018 USENIX Annual Technical Conference* (*USENIX ATC*), July 2018, pp. 951-965.

[40] Yingying Tian, Sooraj Puthoor, Joseph L. Greathouse, Bradford M. Beckmann, and Daniel A. Jiménez. "Adaptive GPU cache bypassing." In *Proceedings of the 8th Workshop on General Purpose Processing using GPUs* (GPGPU). pp. 25-35.

[41] H.-H Lee, G. Tyson and M. Farrens. Eager writeback – a technique for improving bandwidth utilization. *In the proceedings of* the *33rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2000.

[42] J. Stuecheli, D. Kaseridis, D. Daly, H. C. Hunter and L. K. John. The virtual write queue: Coordinating DRAM and last-level cache policies. In the proceedings of *the 37th International Symposium on Computer Architecture (ISCA)*, 2010.

[43] C. Jeon, A. Li, L. Cox and S. Rixner. Reducing DRAM row activations with eager read/write clustering. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), Article 43, 2013.

[44] V. Seshadri, A. Bhowmick, O. Mutlu, P. B. Gibbons, M. A. Kozuch and T. C. Mowry. The dirty-block index. In the proceedings of *the 41st International Symposium on Computer Architecture (ISCA)*, 2014.

[45] Yifan Sun, Trinayan Baruah, Saiful A. Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, Harrison Barclay, Amir Kavyan Ziabari, Zhongliang Chen, Rafael Ubal, José L. Abellán, John Kim, Ajay Joshi, and David Kaeli. "MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization." To appear in *Proceedings of 46th International Symposium on Computer Architecture* (ISCA), June 2019.

[46] Jonathan Lew, Deval Shah, Suchita Pati, Shaylin Cattell, Mengchi Zhang, Amruth Sandhupatla, Christopher Ng, Negar Goli, Matthew D. Sinclair, Timothy G. Rogers, and Tor M. Aamodt. Analyzing Machine Learning Workloads Using a Detailed GPU Simulator. In IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2019.

[47] Matthew D. Sinclair, Johnathan Alsop, and Sarita V. Adve. 2015. Efficient GPU synchronization without scopes: saying no to complex consistency models. In *Proceedings of the 48th International Symposium on Microarchitecture* (MICRO-48). pp. 647-659.

[48] Blake A. Hechtman, Shuai Che, Derek R. Hower, Yingying Tian, Bradford M. Beckmann, Mark D. Hill, Steven K. Reinhardt, and David A. Wood, "QuickRelease: A throughput-oriented approach to release consistency on GPUs," *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Orlando, FL, 2014, pp. 189-200.

[49] Singh, Inderpreet, Arrvindh Shriraman, Wilson WL Fung, Mike O'Connor, and Tor M. Aamodt. "Cache coherence for GPU architectures." In 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), pp. 578-590. IEEE, 2013.

[50] Power, Jason, Arkaprava Basu, Junli Gu, Sooraj Puthoor, Bradford M. Beckmann, Mark D. Hill, Steven K. Reinhardt, and David A. Wood. "Heterogeneous system coherence for integrated CPU-GPU systems." In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 457-467. ACM, 2013.

[51] Pei, Songwen, Myoung-Seo Kim, Jean-Luc Gaudiot, and Naixue Xiong. "Fusion coherence: scalable cache coherence for heterogeneous kilo-core system." In Advanced Computer Architecture, pp. 1-15. Springer, Berlin, Heidelberg, 2014.

[52] Orr, Marc S., Shuai Che, Ayse Yilmazer, Bradford M. Beckmann, Mark D. Hill, and David A. Wood. "Synchronization using remote-scope promotion." In ACM SIGPLAN Notices, vol. 50, no. 4, pp. 73-86. ACM, 2015.

[53] Alsop, Johnathan, Marc S. Orr, Bradford M. Beckmann, and David A. Wood. "Lazy release consistency for GPUs." In The 49th Annual IEEE/ACM International Symposium on Microarchitecture, p. 26. IEEE Press, 2016.

[54] Alsop, Johnathan, Matthew D. Sinclair, and Sarita V. Adve. "Spandex: a flexible interface for efficient heterogeneous coherence." In Proceedings of the 45th Annual International Symposium on Computer Architecture, pp. 261-274. IEEE Press, 2018.

[55] Vijaykumar, Nandita, Abhilasha Jain, Diptesh Majumdar, Kevin Hsieh, Gennady Pekhimenko, Eiman Ebrahimi, Nastaran Hajinazar, Phillip B. Gibbons, and Onur Mutlu. "A case for richer cross-layer abstractions: Bridging the semantic gap with expressive memory." In 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), pp. 207-220. IEEE, 2018.

[56] Li, Chao, Shuaiwen Leon Song, Hongwen Dai, Albert Sidelnik, Siva Kumar Sastry Hari, and Huiyang Zhou. "Locality-driven dynamic GPU cache bypassing." In Proceedings of the 29th ACM on International Conference on Supercomputing, pp. 67-77. ACM, 2015.

[57] Li, Ang, Gert-Jan van den Braak, Akash Kumar, and Henk Corporaal. "Adaptive and transparent cache bypassing for GPUs." In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 17. ACM, 2015.