Virgo: Cluster-level Matrix Unit Integration in GPUs for Scalability and Energy Efficiency

Hansung Kim*, Ruohan Yan*, Joshua You, Tieliang Vamber Yang, Yakun Sophia Shao
University of California, Berkeley
{hansung_kim,yrh,jyou12,vamber,ysshao}@berkeley.edu

Abstract

Modern GPUs incorporate specialized matrix units such as Tensor Cores to accelerate GEMM operations central to deep learning workloads. However, existing matrix unit designs are tightly coupled to the SIMT core, limiting the size and energy efficiency of the operation due to capacity and bandwidth constraints from the register file. Such a limitation in scalability makes it difficult to simultaneously enhance compute throughput and improve energy efficiency in GPUs.

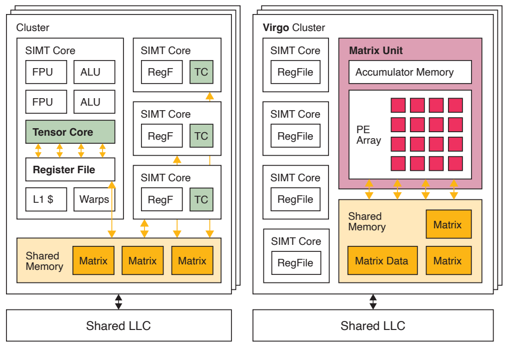
To address this challenge, we propose *Virgo*, a new GPU microarchitecture that integrates matrix units at the *SIMT core cluster* level. By physically disaggregating the matrix unit from the core, Virgo eliminates scalability constraints imposed by the core microarchitecture. Consequently, Virgo enlarges the granularity of operation which not only improves data reuse, but also reduces the number of instructions processed in the core. This decreases energy consumption within the core pipeline, thereby improving the system-level energy efficiency. Our evaluations with synthesized RTL demonstrate that Virgo achieves up to 66.3% reduction in active power and 77.2% reduction in on-chip energy consumption compared to the baseline core-coupled design.

1 Introduction

In recent years, the computational capability of GPUs has surged at an unprecedented rate, driven by the demand of large-scale deep learning applications such as large language models [7, 15, 30]. To meet the demands, modern GPUs have incorporated specialized hardware units such as NVIDIA Tensor Cores [9] and AMD Matrix Cores [1] to significantly accelerate matrix operations prevalent in the applications.

As the demand for compute capabilities continue to grow, so has the scale of integration of matrix units: the FLOPS achieved by Tensor Cores has increased eight-fold from Volta to Hopper in five years [9, 12]. Beyond FLOPS, power and energy have become increasingly important for modern GPU workloads. Deep learning applications on GPUs are known to be not only energy-intensive [19], but also power-constrained, especially in datacenters [6, 24–26, 38].

However, it is increasingly challenging to simultaneously meet the demand for both higher FLOPS and energy efficiency, due to the *tight coupling* between the matrix unit and the GPU SIMT core. Matrix units are typically integrated into



(a) GPU Cluster with Tensor Cores (b) GPU Cluster with Disaggregated Matrix Unit

Figure 1. Overview of (a) today's GPU architecture with tightly coupled integration of Tensor Cores (TC), compared to (b) Virgo's cluster-level integration of matrix units.

the core pipeline as specialized functional units, receiving data through the register file via the standard instruction datapath. This tightly-coupled integration limits the operation size due to the capacity and bandwidth constraints of the register file, with tile sizes such as 16×8×16 [29, 32]. Such a small granularity of operation requires processing a large number of instructions in the core pipeline, consuming substantial energy and power in instruction decoding and scheduling, rather than in actual computation.

To address these challenges, we propose Virgo, a novel GPU architecture that integrates dedicated matrix units at the *cluster* level (Figure 1). The cornerstone of Virgo is the physical disaggregation of the matrix unit from the core microarchitecture, thereby eliminating scalability constraints. As a result, Virgo enables not only to flexibly increase the operation granularity at the hardware level, but also to reduce the number of instructions processed by the core, lowering the overall system-level energy. Our key contributions are:

- We propose a novel cluster-level matrix unit integration for GPUs that enhances scalability and energy efficiency by disaggregating the accelerator from core.
- We fully implement GPU designs featuring both the proposed cluster-level and the baseline core-coupled integration in synthesizable RTL, as well as the software programming interface. The entire Virgo design will be made available as open source.
- We demonstrate that Virgo, synthesized using a commercial 16nm process, improves active on-chip power consumption by 66.3% and energy by 77.2%, compared to the core-coupled baseline.

* These authors contributed equally to this work.

GPU	V100	A100	H100
Architecture	Volta	Ampere	Hopper
Tensor FP16 TFLOPS	1x	2.5x	7.9x
CUDA FP32 TFLOPS	1x	1.2x	4.3x
# of Tensor Cores	1x	0.7x	0.8x
MACs in 1 Tensor Core	**64**	**256**	**473**
Registers per thread (max. 255)	224	221	168
Warp occupancy	12.5%	10.0%	14.1%

Table 1. Scaling trends of the compute capabilities of NVIDIA datacenter GPUs across generations, and their occupancy characterization of CUTLASS GEMM kernels.

2 Background and Motivation

This section provides an overview of the GPU and Tensor Core microarchitectures, highlighting the scalability and efficiency limitations of the current Tensor Core integration.

2.1 GPU Cluster Microarchitecture

Figure 1(a) shows the GPU microarchitecture featuring *clusters* of SIMT cores connected to a shared last-level cache. A cluster houses multiple SIMT cores interconnected to the *shared memory*, a software-managed scratchpad.

The clustered organization of GPU cores offer a key benefit of increased data sharing across threads. With more cores interconnected to the shared memory via the cluster, a greater number of threads can share data through the memory. This is especially important for DNN workloads with large GEMM operations, which make extensive use of the shared memory for tiling and data re-use. Cluster-based architecture is widely adopted in the industry, known as *Streaming Multiprocessors* in NVIDIA [12], *Compute Units* in AMD CDNA [1] and *X^e-cores* in Intel X^e-HPG architecture [20].

2.2 State-of-the-art Tensor Core Integration

To meet the rapidly increasing compute demand of deep learning applications, NVIDIA introduced dedicated matrix units known as *Tensor Cores* in the Volta architecture [9]. Tensor Core consists of multiple SIMD-parallel dot product units designed for high-throughput multiply-add operations [27]. As shown in Figure 1(a), Tensor Core receives matrix operands directly through the register file, essentially being a specialized execution unit tightly coupled into the pipeline of the SIMT core. Notably, as Table 1 shows, Tensor Core has been rapidly growing in its compute capability, not as in the total number of instances, but rather as in the size of each individual instance [9, 10, 12]. This trend underscores the need to ensure scalability of integrating larger-scale matrix units for future generations of GPUs.

2.3 Limitations of the Core-Coupled Approach

However, the tightly core-coupled nature of the Tensor Core design poses significant challenges in scalability. First, modern GEMM kernels generate high *register pressure* as they require extensive use of register file space to store multiple input and accumulator matrix tiles [21]. As a result, this leads to decreased kernel occupancy and frequent register spills in Tensor Core-accelerated GEMM kernels [5, 33, 35–37]. Our characterization of CUTLASS confirms this, where the register usage per thread is very high, leading to low occupancy (Table 1). While occupancy does not always impact performance, it underlines the challenge in further scaling up the size of matrix units given the current capacity constraints. INTERPRET [23] and Duplo [22] attempt to alleviate this constraint by reducing duplicated data in the register file.

Furthermore, tight coupling to the register file imposes not only *capacity*, but also *bandwidth* constraints. The available register file bandwidth is exhausted during Tensor Core operations in Volta [27]. To scale up operation size, an increase in register file bandwidth is necessary to accommodate the larger operand data, posing significant design challenges.

Due to such constraints, core-coupled matrix units support *fine-grained* operations. For example, NVIDIA Tensor Cores support tile sizes of 16×8×16 [13]; AMD CDNA2 Matrix Cores support 16×16×16 and 32×32×8 [29]. Such small tile sizes require processing a large number of instructions to complete the entire GEMM. This results in significant energy expenditure in hardware beyond the matrix unit itself, such as the register file, instruction issue, and warp scheduling.

2.4 Recent Advances and Remaining Challenges

The Hopper Tensor Core addresses register pressure by introducing the new `wgmma` instruction, which can read matrix operands directly from shared memory, reducing reliance on the register file [12, 25]. This new method of operand delivery obviates the need to store the input matrix in the register file, alleviating the scalability limitation.

However, the Hopper Tensor Core does not fully solve the register pressure issue, and energy efficiency remains a concern. The `wgmma` instruction still requires storing the result matrix back to the register file [13]; register pressure remains significant (Table 1). Additionally, the Tensor Core is still tightly bound to the core [12, 25], preventing data sharing between units. This limitation restricts data reuse compared to a large unified design, reducing energy efficiency.

Virgo addresses the scalability and energy efficiency challenges by completely disaggregating the matrix unit from the cores, establishing it as a separate instance at the cluster level. To the best of our knowledge, Virgo is the first to integrate a matrix engine at this level. Virgo's matrix unit, drawing operand directly from shared memory and storing results in its own accumulator memory, bypasses the register file completely, eliminating the capacity and bandwidth constraints. Moreover, because the hardware can operate on larger matrix operands, the granularity of operation increases, reducing the number of instructions in the core and thereby cutting down power and energy consumption.

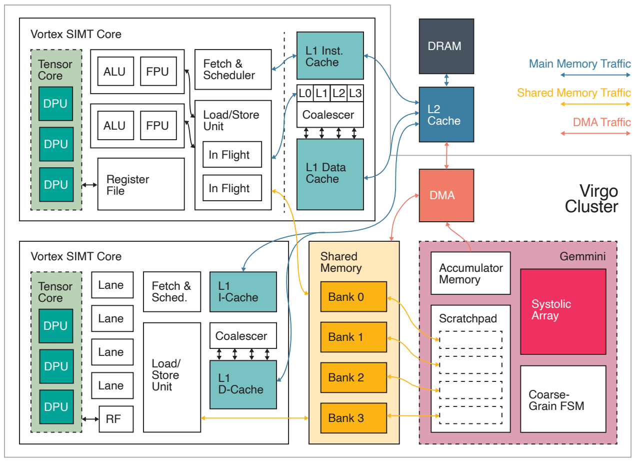


Figure 2. Microarchitecture of a Virgo cluster. The Gemmini-based matrix unit is disaggregated from the SIMT cores into a separate unit, being supplied its operand through the shared memory. Dashed lines indicate optionally instantiated modules for evaluation, such as the core-coupled Tensor Core.

3 Virgo Microarchitecture

Virgo aims to improve scalability and energy efficiency by disaggregating the matrix unit from the core. We discuss the key components for realizing this design: (1) a command interface through which the core and the matrix unit can communicate, (2) a shared memory interconnect that efficiently handles concurrent accesses from the core and the matrix unit, (3) efficient matrix data movement via a dedicated DMA engine, and (4) a synchronization mechanism.

We first mention key open-source hardware we leverage to realize our design at the RTL. First, we use Vortex [34], an open-source GPGPU implementation that enables full stack development of GPU hardware and software in the RISC-V ISA. Second, we use Gemmini [17], a Chisel-based systolic array generator, to generate the matrix unit for Virgo[1].

3.1 Command Interface to the Matrix Unit

We facilitate the cluster-local memory interconnect to establish an efficient MMIO-based command interface between the core and the matrix unit, without the need to modify the Vortex RISC-V ISA. Specifically, we replace the RoCC interface [3] of Gemmini with a new MMIO interface. To perform a matmul operation, the core writes to a control register mapped to the shared memory address space, which triggers the Gemmini unit to retrieve matrix operands from the shared memory, iterate through the $i/j/k$ dimensions, and compute a single 64×64×64 GEMM tile via the systolic array. To synchronize upon completion, a control register exposes the busy signal of the matrix unit which the core polls in a loop at the cluster-wide barrier.

[1]Using *Vortex* and *Gemmini* to realize *cluster*-level integration motivates the name *Virgo*.

3.2 Memory System

Key challenges of Virgo's memory system includes handling concurrent, heterogeneous accesses to the shared memory from both the core and the matrix unit, and achieving high throughput in matrix data delivery from the global memory.

3.2.1 Shared memory interconnect.

The SIMT lanes of a Vortex core issues 32-bit word-sized memory accesses, while the Gemmini-based matrix unit makes much wider accesses to supply the $n \times n$ systolic array. Moreover, the core and the matrix unit frequently perform opposite read and write operations in a producer-consumer relationship as discussed in Section 4.3.2. To tackle these challenges, we choose to:

- *Support for both wide/narrow requests.* Gemmini's wide memory requests are split into word-sized sub-requests distributed across the SRAM banks. This allows the SRAM banks to be able to serve both narrow requests from the SIMT lanes and wide requests from the matrix unit with minimal bank conflicts.
- *Two-dimensional banking*. The shared memory SRAM is banked not only *horizontally* in word sizes, but also *vertically* in line sizes to allow bank-level parallelism for both SIMT core requests and Gemmini requests.
- *Separate read and write channels.* To efficiently support concurrent read/write accesses from the core and the matrix unit, we use separate memory channels for read and write requests.

We make extensive use of TileLink [8] and Diplomacy [31], enabling a highly parameterized design for different number of cores, matrix units, and memory widths in the cluster.

3.2.2 Efficient Data Movement with DMA.

We find that using regular loads and stores in the SIMT core for moving matrix data from global to shared memory significantly limits the matrix unit utilization (Table 2), due to both address computation overhead and low instruction throughput of the Vortex core. We therefore incorporate a programmable DMA engine to facilitate efficient data move of the matrix tile between the global memory and shared memory, as depicted in Figure 2. Note that the DMA can also be incorporated to the baseline core-coupled design, which we also evaluate in order to model an Ampere-like design that features both the Tensor Core and an asynchronous data copy engine [11].

3.3 Cluster-wide Synchronization

As will be discussed in Section 4.2, the SIMT cores in a cluster collaborate together to move matrix data, or do post-processing computation on the result from the unit. This requires an efficient synchronization mechanism *across* cores in the same cluster. To this end, we design a synchronizer module that interfaces with the warp scheduler of the cores to synchronize the progress of warps cluster-wide. We reuse Vortex's `vx_bar` instruction to allow the programmer to specify which warps participate in the cluster-wide barriers.

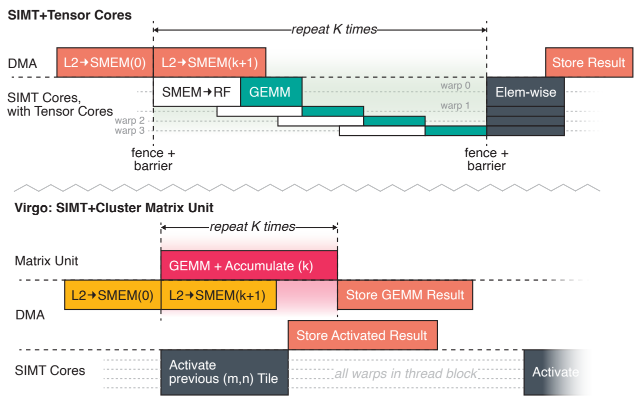


Figure 3. Asynchronous execution scheme of the matrix unit and SIMT cores in Virgo during a GEMM operation, compared to the Tensor Core-based design.

4 Virgo Programming Model

As a result of increased scalability, Virgo's cluster-level matrix unit operates on larger tile sizes, which results in a longer operation latency. While SIMT cores employ warp multithreading to hide the latency of common instruction operations, multithreading alone is not sufficient to completely hide the latency of matrix operations. Therefore, Virgo provides an *asynchronous* programming interface to allow the programmer to maximally utilize compute resources in both the core and the matrix unit.

4.1 Asynchronous Programming of Matrix Unit

As discussed in Section 3.1, the Gemmini-based matrix unit exposes a memory-mapped IO interface to the core. The MMIO accesses are designed to be non-blocking, which allows the core to start matrix unit execution asynchronously and then proceed to the next instruction. For instance, after initiating compute of a tile, the core can then proceed to data movement of the next tile. This will be detailed in Section 4.3.

Virgo also allows for instantiating *multiple* Gemmini units at each cluster, in which case Virgo establishes a separate multiple MMIO interface at a non-overlapping address region for each unit, along with a unique ID. The programmer can then fully manage partitioning work onto multiple matrix units by using the ID as a handle in the kernel.

4.2 Collaborative Execution of Warps

Before or after the matrix unit executes an operation, the matrix elements are re-mapped to the SIMT threads in the core for pre- or post-processing, such as data movement to off-chip memory or element-wise activations in a DNN. Because Virgo's matrix unit operates on a larger tile size than the core-coupled designs, *multiple* warps collaborate to participate in a *single* matrix operation, depicted in Figure 3. This collaborative execution scheme necessitates the cluster-wide synchronization mechanism described in Section 3.3.

4.3 Mapping to the GEMM Kernel

We now describe how the the aforementioned mechanisms can be utilized to implement an optimized GEMM kernel.

4.3.1 Thread block tiling. Virgo builds upon the well-established multi-level tiling scheme of GEMM kernels on GPUs [21]. A key difference is that while Tensor Cores compute *warp* tiles, Virgo accelerates the entire *thread block* tile in hardware. Each thread block, spatially partitioned across the (M,N) output space, iterates over the inner (K) dimension in a temporal loop, as in Figure 3. As the loop iterates, the Gemmini unit accumulates partial sum data onto its private accumulator memory. Multiple thread blocks compute multiple (M,N) output tiles in parallel.

4.3.2 Software pipelining and double buffering. Importantly, the Virgo-optimized GEMM kernel employs *software pipelining* which is enabled by the asynchronous programming interface of the matrix unit. As shown in Figure 3, while the matrix unit is computing a tile along the K dimension consisting a *consumer* pipeline, either the DMA unit or a group of SIMT core warps collaboratively fetch the next input tile along the K dimension from the global memory to the shared memory, consisting the *producer* pipeline. Another set of SIMT core warps can collaborate to form an additional consumer pipe that computes the activation function on a previous result tile. Because both the producer and consumer pipes run in parallel, the tile data are double-buffered in the shared memory. This mechanism allows all of the SIMT core, the matrix unit and the DMA to be maximally utilized throughout the kernel.

5 Methodology

5.1 Implementation

We use the Chipyard SoC generator framework [2] to integrate all hardware components into a system-on-chip.

Vortex SIMT Cores. We derive the SIMT core and L1 cache from Vortex and integrate them into the Chipyard SoC. We make several improvements to the RTL including a memory coalescer, memory store fences, and optimizing the warp scheduler. We find these changes are necessary to allow us to focus on bottlenecks that pertain to the different integration methods of the matrix unit.

Gemmini. We make extensive modifications to Gemmini, detailed in Sections 3.1 and 3.2.1.

Tensor Cores. To faithfully model the baseline GPU with core-coupled matrix units, we closely implement in RTL the microarchitecture of Volta Tensor Core proposed in [27], whose timing behavior is correlated to the NVIDIA GPU. To account for the fewer SIMT lanes in our Vortex configuration (8) than in Volta (32), we instantiate a single octet instead of four [27]. In addition, since the Vortex core lacks support for FP16, we halve the K dimension to fit the FP32 operand tile in the available register file bandwidth of 1.5kbits/s. As a result,

	256×256×256		128×512×512		512×512×512	
	Cycle	%Util	Cycle	%Util	Cycle	%Util
TC	726k	36.1	1450k	36.2	5.79M	36.2
TC + DMA	458k	57.2	832k	63.0	3.34M	62.7
Virgo	540k	48.5	941k	55.7	3.71M	56.5
Virgo + DMA	**310k**	**84.5**	**583k**	**90.0**	**2.30M**	**91.0**

Table 2. Execution time and % utilization of the MAC units in Virgo and baseline core-coupled Tensor Core designs ("TC") for GEMM kernels of three different dimensions.

the operation size of a wmma instruction is (m,n,k)=(8,8,8), scaled down from (16,16,16) in Volta. We use 0x7B opcode reserved in the ISA for wmma instruction.

Through microbenchmarks, we verify that our Tensor Core implementation achieves identical timing behavior of 2 cycles per operation as in Volta, with identical compute throughput of 512bits (= 4×4×2 FP16 = 4×2×2 FP32) per octet per cycle. Both Tensor Core and Gemmini use HardFloat [18] floating point units for fair area/power comparison.

5.2 Experiments

Configuration. Both Virgo and core-coupled baseline designs are configured to 1 cluster with 4 cores, each with 8 threads/8 warps. Shared memory is 64KB. Gemmini is configured to 8×8 systolic array with 16KB accumulator memory. The Gemmini unit has the same number of MACs (64) as the 4 Tensor Cores in the baseline design for fair comparison.

Measurement. We synthesize the design using a commercial 16nm process, and use Cadence Joules to measure power from RTL simulation. The Virgo design consumes 3.6% larger SoC area than the core-coupled design at 400MHz.

For all measurements, we discuss *active power*, obtained by subtracting the power of a fully idle design. We find this is necessary for meaningful discussion, as otherwise the SIMT core power is unrealistically high even at idle, possibly due to absence of clock gating. By discussing active power, we are able to distinguish the power implications that arise from our design choices from those that depend on implementation.

6 Evaluation

6.1 GEMM Kernels

We first evaluate GEMM kernels, a fundamental workload that underlies modern deep learning applications on the GPU. The kernels are written separately for the Virgo and core-coupled design with bespoke optimizations, including double-buffering, warp specialization and persistent threads [21].

6.1.1 Performance & Utilization. Table 2 lists cycle count and MAC hardware utilization across three GEMM dimensions. Virgo shows higher hardware utilizations compared to the core-coupled Tensor Cores across all GEMM dimensions, both with or without DMA. The higher utilization of Virgo is mainly attributed to a coarser unit of operation of the matrix unit, compared to the much finer-grained instructions for

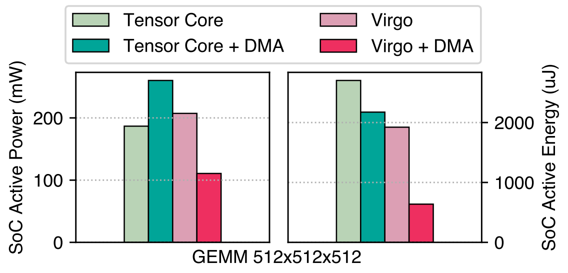


Figure 4. Active power and energy comparison between Virgo and the baseline Tensor Core design, with and without incorporation of DMA.

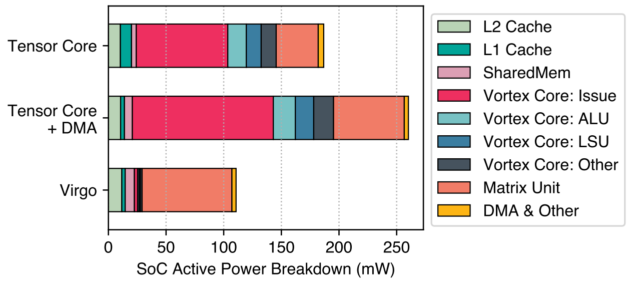


Figure 5. Active power breakdown by SoC components. The core active power is reduced significantly as a result of reduced instruction processing in Virgo. Matrix unit power is higher in Virgo due to higher MAC utilization.

the Tensor Core. As a result, the Tensor Core utilization is constrained by the instruction throughput available from the SIMT core. We note that the utilization numbers for the core-coupled design, although lower than Virgo, remains within range of what is reported in other works [15, 36].

For both Virgo and core-coupled designs, DMA appears to be critical for high utilization. Without DMA, the kernel relies on load/store instructions for data delivery with insufficient throughput limited by instruction issue rate, resulting in matrix units being starved for operand data.

6.1.2 GEMM Power & Energy. In Figure 4 we show a side-by-side comparison between the active power usage of Virgo and core-coupled Tensor Core based designs. Comparing both designs with DMA, Virgo reduces active power by 57.4%. Combined with higher utilization of the matrix unit, energy reduction is even higher, at 70.7%.

Larger operation granularity saves core power. We give detailed power breakdowns at the SoC (Figure 5) and the matrix unit (Figure 6). Importantly, the most reduction in power happens in the core rather than in the matrix unit. Since the core-coupled Tensor Core has a much *finer* granularity of operation, it requires the core to process a larger volume of instructions, consuming significant power in the issue pipeline stage and increasing system power. On the

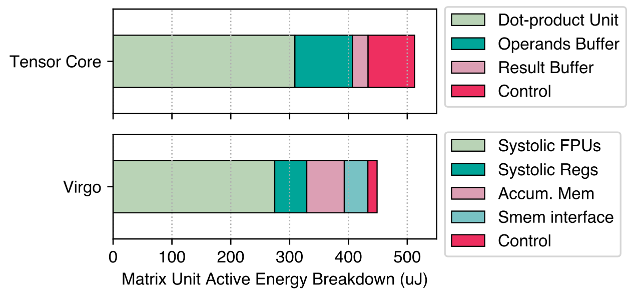


Figure 6. Active energy breakdown of the matrix unit for the 512×512×512 GEMM kernel.

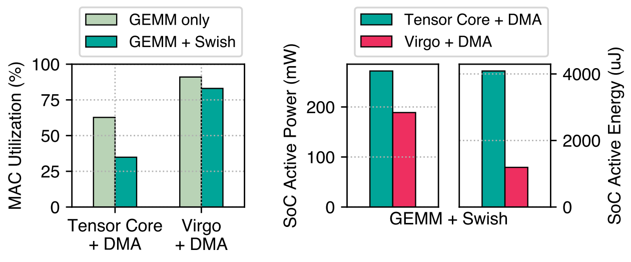


Figure 7. Left: MAC utilization of the fused GEMM 512×512 and activation kernel, compared to GEMM-only. Right: Active power and energy of the fused kernel.

contrary, Virgo achieves a much lower system power consumption by *coarsening* the granularity of operation and minimizing the incidental power consumed outside of the matrix unit. The core pipeline activity measurement supports this argument: The number of retired instructions in Virgo is 8.2% of that of the Tensor Core design.

Matrix unit energy. Figure 6 shows a detailed breakdown of energy consumption inside the matrix units. Although the two designs consume similar amounts of energy in the floating-point units, Virgo uses less energy in its registers. Tensor Core requires FIFO queues with sufficient depth to buffer operand and result data and absorb backpressure from the core pipeline to perform well. Combined with the control logic to manage the buffers and PE stalls, the in-unit energy is higher. Moreover, Virgo's matrix unit is a systolic array design unified at the cluster level, unlike Tensor Cores, which are dot-product units distributed across SIMT cores. This enables Virgo to facilitate better data reuse and use less energy from register accesses. However, Virgo makes use of a sophisticated sharedmem interface and private accumulator memory, offsetting some of the energy savings.

We note that the energy reduction in the SIMT cores that we discussed earlier is a greater factor than that in the matrix units. The choice of systolic array for the matrix unit is not crucial to achieving Virgo's energy efficiency.

6.2 Fused GEMM and Activation Kernel

Modern deep learning workloads incorporate kernel fusion to minimize off-chip memory accesses [14, 16]. Virgo enables efficient mapping of fused kernels with concurrent execution of the matrix unit and the SIMT core. We implement a fused kernel where the matrix unit and SIMT floating-point units are software-pipelined to execute GEMM and element-wise Swish activations [28] respectively, through asynchronous programming for Virgo, and warp-specialization for Tensor Core [4]. Figure 7 shows the loss in MAC utilization is much smaller for Virgo than the core-coupled design, compared to executing GEMM only. In Virgo, matrix unit execution is asynchronous, leaving the warps in the core to participate in non-matrix work without stalls. However, Tensor Core instructions are synchronous, necessitating warp specialization to overlap matrix and non-matrix operations. This results in fewer warps participating in each operation, and thus overall lower hardware utilization.

6.3 Multiple Heterogeneous Matrix Units

The architectural disaggregation of the matrix unit at the cluster level enables a new design space in Virgo, where *multiple* matrix units with heterogeneous configurations are integrated to the cluster. We demonstrate this by instantiating two matrix units with different systolic dimensions: 8×8 and 4×4, and executing two different sized GEMMs on the units in the kernel: 256×256×256 and 128×128×256. We verify that compared to the case where both GEMMs run serially on a single 8×8 unit, the hardware utilization of the two matrix units (59.5%) is consistent with the utilization of the single matrix unit (59.7%). This indicates that Virgo cluster's memory architecture scales well to more matrix units, demonstrating the scalability of the design.

7 Conclusion

We present *Virgo*, a novel GPU architecture that integrates matrix units at the cluster level. Virgo solves the scalability and energy efficiency challenges faced by the core-coupled designs in current GPUs, by physically disaggregating the matrix unit from the core into a separate unit at the cluster. This disaggregation obviates the need to deliver operand data solely from the register file, eliminating capacity and bandwidth constraints. As a result, Virgo enables larger-scale matrix operations at hardware, which not only improves data reuse, but also reduces the number of instructions processed in the core, improving overall system-level energy efficiency.

We implement our design in RTL by leveraging open-source RISC-V infrastructures such as Vortex and Gemmini, and demonstrate significant improvement in power and energy efficiency. We also show new ways to map applications with Virgo, enabling concurrent and collaborative execution between the matrix unit and the SIMT core, or integrating multiple matrix units with different configurations.

References

[1] AMD. Amd cdna 2 architecture. https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf, 2021.

[2] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[3] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 4:6–2, 2016.

[4] Michael Bauer, Henry Cook, and Brucek Khailany. Cudadma: optimizing gpu memory bandwidth via warp specialization. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, pages 1–11, 2011.

[5] Ganesh Bikshandi and Jay Shah. A case study in cuda kernel fusion: Implementing flashattention-2 on nvidia hopper architecture using the cutlass library. *arXiv preprint arXiv:2312.11918*, 2023.

[6] Robert A Bridges, Neena Imam, and Tiffany M Mintz. Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Computing Surveys (CSUR)*, 49(3):1–27, 2016.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Henry Cook. *Productive Design of Extensible On-Chip Memory Hierarchies*. PhD thesis, EECS Department, University of California, Berkeley, May 2016.

[9] NVIDIA Corporation. Nvidia tesla v100 gpu architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2017.

[10] NVIDIA Corporation. Nvidia a100 tensor core gpu datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf, 2022.

[11] NVIDIA Corporation. Nvidia ampere gpu architecture tuning guide. https://docs.nvidia.com/cuda/pdf/Ampere_Tuning_Guide.pdf, 2024.

[12] NVIDIA Corporation. Nvidia h100 tensor core gpu datasheet. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet, 2024.

[13] NVIDIA Corporation. Parallel thread execution isa version 8.5. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html, 2024.

[14] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[15] Michael Davies, Ian McDougall, Selvaraj Anandaraj, Deep Machchhar, Rithik Jain, and Karthikeyan Sankaralingam. A journey of a 1,000 kernels begins with a single step: A retrospective of deep learning on gpus. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 20–36, 2024.

[16] Roman Dubtsov, Evarist Fomenko, and Babak Hejazi. New cublas 12.0 features and matrix multiplication performance on nvidia hopper gpus, 2023. NVIDIA Technical Blog.

[17] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert Ou, Max Banister, et al. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv preprint arXiv:1911.09925*, 3:25, 2019.

[18] John Hauser. Berkeley hardfloat. http://www.jhauser.us/arithmetic/HardFloat.html, 2019.

[19] Miro Hodak, Masha Gorkovenko, and Ajay Dholakia. Towards power efficiency in deep learning on data center hardware. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1814–1820. IEEE, 2019.

[20] Intel. Introduction to the xe-hpg architecture. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-the-xe-hpg-architecture.html, 2022.

[21] Andrew Kerr, Duane Merrill, Julien Demouth, and John Tran. Cutlass: Fast linear algebra in cuda c++. *NVIDIA Developer Blog*, 2017.

[22] Hyeonjin Kim, Sungwoo Ahn, Yunho Oh, Bogil Kim, Won Woo Ro, and William J Song. Duplo: Lifting redundant memory accesses of deep neural networks for gpu tensor cores. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 725–737. IEEE, 2020.

[23] Jae Seok Kwak, Myung Kuk Yoon, Ipoom Jeong, Seunghyun Jin, and Won Woo Ro. Interpret: Inter-warp register reuse for gpu tensor core. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 309–319. IEEE, 2023.

[24] Baolin Li, Rohin Arora, Siddharth Samsi, Tirthak Patel, William Arcand, David Bestor, Chansup Byun, Rohan Basu Roy, Bill Bergeron, John Holodnak, et al. Ai-enabling workloads on large-scale gpu-accelerated system: Characterization, opportunities, and implications. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1224–1237. IEEE, 2022.

[25] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. Benchmarking and dissecting the nvidia hopper gpu architecture. *arXiv preprint arXiv:2402.13499*, 2024.

[26] Pratyush Patel, Zibo Gong, Syeda Rizvi, Esha Choukse, Pulkit Misra, Thomas Anderson, and Akshitha Sriraman. Towards improved power management in cloud gpus. *IEEE Computer Architecture Letters*, 22(2):141–144, 2023.

[27] Md Aamir Raihan, Negar Goli, and Tor M Aamodt. Modeling deep learning accelerator enabled gpus. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 79–92. IEEE, 2019.

[28] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[29] G Schieffer, D Medeiros, J Faj, A Marathe, and I Peng. Characterizing the performance, power efficiency, and programmability of amd matrix cores. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), 2024.

[30] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[31] Henry Cook SiFive. Diplomatic design patterns : A tilelink case study. 2017.

[32] Wei Sun, Ang Li, Tong Geng, Sander Stuijk, and Henk Corporaal. Dissecting tensor cores via microbenchmarks: Latency, throughput and numeric behaviors. *IEEE Transactions on Parallel and Distributed Systems*, 34(1):246–261, 2022.

[33] Guangming Tan, Linchuan Li, Sean Triechle, Everett Phillips, Yungang Bao, and Ninghui Sun. Fast implementation of dgemm on fermi gpu. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2011.

[34] Blaise Tine, Krishna Praveen Yalamarthy, Fares Elsabbagh, and Kim Hyesoon. Vortex: Extending the risc-v isa for gpgpu and 3d-graphics. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 754–766, 2021.

[35] Jiajun Wang, Ahmed Khawaja, George Biros, Andreas Gerstlauer, and Lizy K John. Optimizing gpgpu kernel summation for performance and energy efficiency. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 123–132. IEEE, 2016.

[36] Da Yan, Wei Wang, and Xiaowen Chu. Demystifying tensor cores to optimize half-precision matrix multiply. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 634–643. IEEE, 2020.

[37] Xiuxia Zhang, Guangming Tan, Shuangbai Xue, Jiajia Li, Keren Zhou, and Mingyu Chen. Understanding the gpu microarchitecture to achieve bare-metal performance tuning. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 31–43, 2017.

[38] Dan Zhao, Siddharth Samsi, Joseph McDonald, Baolin Li, David Bestor, Michael Jones, Devesh Tiwari, and Vijay Gadepally. Sustainable supercomputing for ai: Gpu power capping at hpc scale. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pages 588–596, 2023.