Granularity- and Interference-Aware GPU Sharing with MPS

Alex Weaver*, Krishna Kavi*, Dejan Milojicic†, Rolando Pablo Hong Enriquez†,
Ninad Hogade†, Alok Mishra† and Gayatri Mehta*
*University of North Texas, Denton, Texas
†Hewlett Packard Labs, Milpitas, California

***Abstract*—GPU acceleration has become central to achieving exascale performance for high-performance computing (HPC) applications. Even at this extreme scale, most scientific and HPC-scale DNN applications under-utilize GPU resources. Existing NVIDIA GPU sharing mechanisms—namely, CUDA Multi-Process Service (MPS)—can be used to increase utilization, throughput, and energy efficiency. However, naively co-scheduling workflows often does not yield optimal results. Scheduling multiple workloads with high resource utilization on the same set of GPUs, for example, leads to performance degradation due to resource contention. In short, GPU sharing must be granularity- and interference-aware to maximize the benefit of co-scheduling. We propose a scheduling approach that optimizes workflow scheduling configurations to maximize given system metrics–i.e., throughput and energy efficiency; uses workload profiling data to right-size GPU resources for combinations of HPC workflows; and collocates workflows using existing concurrency/sharing mechanisms. We show that choosing the right arrangement of workflows to collocate can increase throughput by as much as 2x and energy efficiency by 1.6x using CUDA MPS on NVIDIA A100X GPUs.**

I. Introduction

The ever-growing demand for modern scientific calculations is the underlying factor behind the race to build impressive supercomputers and high performance computing (HPC) systems [1]. Although traditionally, HPC setups have been mainly driven by powerful CPU architectures, the inescapable breakdown of Dennard scaling has fueled the interest in alternate solutions. Initially, this interest was mainly devoted to multicore processes [2] and more recently to the development of different types of accelerators [3]. While there is a tendency to accelerator diversification pointing to a heterogeneous accelerator era in HPC [4], GPUs are still the dominant choice [5]. Therefore, unless there is a significant disruption in the accelerator market, it is essential to find best ways to run HPC workloads on GPUs. In this work, we propose a scheduling approach that incorporates profiling data and GPU-sharing capabilities into workflow schedulers. The key contributions of our work presented here are:

- Evaluation framework for workflow utilization analysis and a scheduling approach that incorporates granularity- and interference-awareness using currently available GPU sharing mechanisms.
- An approach for right-sizing GPU resource allocation using existing MPS sharing mechanism.
- A repository of bare-metal HPC benchmarks that can be run on small prototype HPC clusters in reasonable time-frames and incorporates easy scaling of resources and problem size.

The key takeaways from our analysis that we discuss in detail in Section V are:

- Sharing GPUs between low-utilization applications minimizes interference and yields greater benefits to both system throughput and energy efficiency.
- The relative priority of specific metrics like system throughput and energy efficiency determines the optimal scheduling configuration of workflow tasks.
- Throughput and energy efficiency are impacted differently by the total number of MPS clients scheduled concurrently.

The rest of the paper is organized as follows: In section II we provide relevant background about NVIDIA GPU architecture, which is the accelerator used in our experiments. We also describe the various concurrency mechanisms in these GPUs as well as the intro to GPU utilization and Warp occupancy measurements using the GPUs NVIDIA toolchain.

II. Background

A. GPU Architecture Overview

In this work, we focus exclusively on NVIDIA GPUs, leaving evaluation on AMD and other architectures for future work. The architecture of NVIDIA GPUs is built around the Single-Instruction, Multiple-Thread paradigm, which takes advantage of thread-level parallelism through simultaneous hardware multi-threading. The streaming multiprocessor (SM) creates, manages, schedules, and executes threads. Threads are partitioned by SMs into *warps*, which execute one common instruction at a time for all threads in the warp. Because the individual threads are still free to branch and execute independently, full efficiency is only realized when all threads of a warp agree on their execution path. [6]

B. GPU Concurrency Mechanisms

NVIDIA provides multiple off-the-shelf mechanisms for overlapping kernels from different processes on the same GPU. By default, shared GPU access is provided through a time-sliced scheduler. In **time-slicing**, work from queues belonging to different kernels do not execute concurrently. Instead, they must be swapped in and out of the GPU as each process is scheduled. Additional concurrency mechanisms have been designed to eliminate the overhead of context-switching when sharing GPUs between multiple processes (or kernels).

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00203

CUDA Streams [7] is a feature of the CUDA programming model that provides multiple streams of execution within a single process. Different work from an application can be submitted to independent work queues that are processed concurrently by the GPU. Because streams are part of the same application, they share the same address space and there is no SM performance isolation.

CUDA Multi-Process Service (MPS) [8] allows multiple processes to share compute resources on a single GPU, whether they are cooperative MPI processes or processes from completely separate applications. MPS consists of a control daemon, a server process, and a **client** runtime for each concurrent process. While this mechanism provides memory protections and logical SM partitions between processes—for up to 48 clients—memory bandwidth, scheduling hardware, caches, and memory capacity are shared equally between all client processes.

CUDA Multi-Instance GPU (MIG) [9] is a new feature on NVIDIA GPUs starting with the Ampere architecture that enables hardware partitioning of GPUs into up to 7 separate instances. Each MIG instance has a separate and isolated path through the entire memory system allowing for complete partitioning of memory and compute resources between separate applications. MIG is less flexible than MPS in that it requires a GPU to be idle before the partition configurations can be changed. However, MIG offers much better isolation than MPS.

Fine-grained kernel scheduling. Recent works on fine-grained scheduling [10], [11] have proposed frameworks using the streams sharing mechanism to schedule kernels from multiple applications without context switching overhead. These proposals are geared towards deep learning workloads that switch frequently between compute and memory kernels, leading to increased idle time for GPU resources. The primary aim of these solutions is to maintain low latency for priority applications and address long tail latencies caused by MPS.

C. GPU Utilization and Warp Occupancy

GPU compute utilization is determined not only by the number of active SMs but also by the warp occupancy of SMs during workload execution as well as the percentage of execution time where the GPU is idle. Occupancy is calculated at the kernel level of execution. Each workflow task is comprised of multiple kernels. **Theoretical occupancy**—the upper bound of active warps in an SM—depends on both kernel launch configuration and device capabilities. Limiting factors for theoretical occupancy include total warps, blocks, registers, and shared memory per SM. **Achieved occupancy**—calculated kernel by kernel—describes the actual average percentage of available resources utilized during kernel execution and depends on factors like load balancing and number of blocks launched by the kernel. [6]

III. Motivation

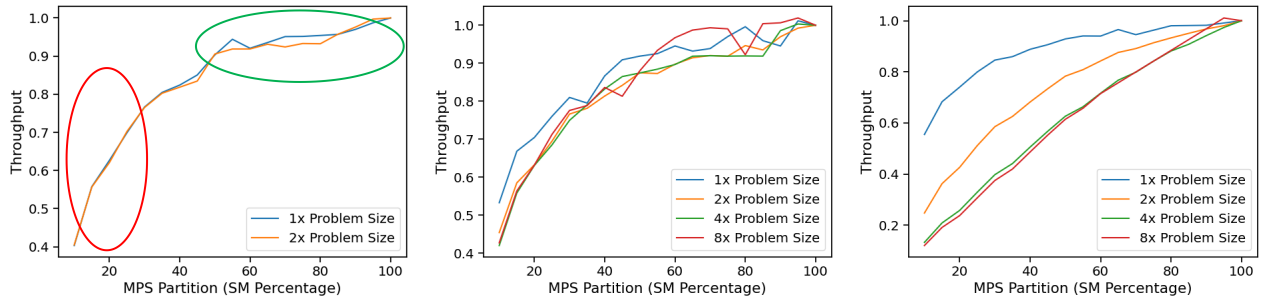
Even in top supercomputers, most GPU-accelerated applications still greatly under-utilize the available GPU resources [12]. On the client side, applications are often optimized for minimal latency rather than GPU utilization. At the same time, the infrastructure provider is motivated to maximize the resource utilization of its hardware to increase overall throughput and energy efficiency. Table I shows both the theoretical and achieved occupancy for selected benchmark test problems. Certain applications like Lammps, which utilizes over 90% of the maximum warps available to it, are unsuited to GPU sharing with MPS because the resulting resource contention will likely degrade overall performance. Instead, other sharing strategies like the one proposed by Orion [11] work better.

A. HPC Workflows

Workload analysis from the National Energy Research Scientific Computing Center (NERSC) suggests that although the programming codes used in these HPC calculations are widely diversified, they tend to represent a reduced number of algorithms [13]. On the other hand, although a significant portion of these codes are already running on GPUs, there are still over 50% of these applications that need performance optimization in either moderate or massive ways. Yet performance is not the only concern, energy consumption is a factor as well [14]. Although GPUs are performance-per watt efficient, given their increasing adoption in HPC systems, it is still imperative to optimize the energy consumption of GPUs alongside their performance [15]. GPU sharing is one way to increase performance and improve energy efficiency. HPC workflows typically consist of multiple sequential and concurrent tasks. Such is the case for several simulation tasks in condense matter or electronic structure calculations (e.g., Lammps [16], Berkeley GW Epsilon [17]) or deep learning workloads in general [18]. As introduced in Section II, MPS allows NVIDIA GPUs to be shared by tasks from separate workflows. Figure 1 shows the change in task throughput as the MPS SM partition is increased from 10% to 100%, demonstrating GPU under-utilization for common workflows benchmarks (i.e., BerkeleyGW-Epsilon [17], Kripke [19], and WarpX [20]). Throughput increases non-linearly in all three

Benchmark	Average Achieved Warp Occupancy	Average Theoretical Warp Occupancy	% of Theoretical Achieved
AthenaPK	13.3%	51.32%	25.92%
BerkeleyGW-Epsilon	23.97%	41.67%	57.52%
Cholla-Gravity	31.45%	37.5%	83.87%
Kripke	32.61%	43.63%	74.74%
Cholla-MHD	17.72%	19.32%	91.72%
LAMMPS	32.7%	35.0%	93.43%
WarpX	24.81%	92.55%	26.81%

TABLE I: Warp occupancy metrics for each benchmark with 2 GPUs and 1x problem size



(a) BerkeleyGW-Epsilon Throughput (b) Kripke Throughput (c) WarpX Throughput

Fig. 1: Change in throughput for three benchmarks as MPS SM partition percentage increases

plots, which suggests decreasing benefit from larger MPS partitions. The green circle in Figure 1a highlights the MPS partition sizes that will yield the best throughput for the BerkeleyGW-Epsilon benchmark, and the red circle highlights partition sizes that will negatively impact workload performance. Figure 1b shows a similar non-linear shape for all input scales for the Kripke benchmark. Finally, we can see that the scale of the input influences the shape of the graph: Larger problem sizes in Figure 1c show more linearity in the partition size-throughput relationship, suggesting larger partition sizes are more beneficial. In summary, the granularity of SM partition size impacts the potential benefit of workflow multi-tenancy using MPS.

IV. Our Scheduling Approach

Our scheduling approach takes into account the GPU memory and compute utilization as well as the power profile of each workload in a workflow and collocates workflows to maximize either throughput or energy efficiency depending on system requirements.

A. Workload Profiling and Evaluation Framework

The first step in our interference- and granularity-aware scheduling is offline profiling of individual workflow tasks. The profiling data gathered in this step is used to predict interference between workflows at the level of a complete workflow task. While other works propose interference-aware collocation at the kernel level of granularity [10], [11], our approach is designed to use off-the-shelf scheduling mechanisms with minimal profiling overhead. We collect GPU compute, memory, and memory bandwidth utilization as well as average power and GPU idle time using the NVIDIA Nsight Systems tool [21] and NVIDIA System Management Interface (SMI) GPU metric query utility. Both of these tools add minimal overhead, and therefore offline profiling only requires the time it takes to run a workflow task. Additionally, because scaling is well-understood for a vast majority of HPC codes, it is possible to infer the utilization characteristics of larger problem sizes from profiling information gathered with smaller workloads. For all workloads we evaluated, the scaling we observed aligned with previously reported data.

B. Optimized Workflow Collocation

Our scheduling approach assumes a pre-existing queue of workflows to be scheduled. The suite of HPC benchmark codes we compiled and evaluated are often used as tasks in larger simulation workflows, in which case an entire queue of workflow tasks as well as data dependencies between them is known before workflow execution. Using the resource utilization information collected in the profiling step, we select groups of queued workflows to co-schedule on available GPUs based on (1) predicted interference and (2) evaluation metric priority (i.e., optimize throughput or energy). *Two workflows are predicted to interfere if they have combined average SM utilization over 100%, combined average memory bandwidth utilization over 100%, or combined maximum memory utilization above the device memory capacity.* In general, the co-scheduling decision is based on the following criteria:

1) Workflows with the lowest compute utilization are prioritized for co-scheduling over high-utilization workflows.
2) Total compute utilization is kept under 100% for all scheduled workflows combined.
3) Combined maximum memory utilization must be kept under GPU device memory capacity. Because scheduling takes place at the level of workflow tasks and not GPU kernel, we take into account the maximum memory requirement for each task.
4) If energy efficiency is prioritized, the maximum number of MPS clients available are used. Otherwise, if throughput is prioritized, the number of clients is limited to 2.

C. Evaluation Metrics

One advantage of this approach to co-scheduling is that we can prioritize different metrics depending on the requirements of the system. In our evaluation—Section V, we show the trade-off between system throughput and total energy efficiency. **Energy efficiency** is measured by the reduction in total GPU energy with MPS over sequential scheduling—i.e., jobs are scheduled individually on GPUs in queue order with no parallel overlap. **Throughput** is defined as the number of tasks completed in a given time and is also calculated relative to sequential scheduling. Finally, a **product metric** gives priority to either throughput or energy efficiency similar to the Energy-Delay product used in the computer architecture discipline.

Benchmark	Problem Size	Max Memory (MiB)	Avg Memory BW Utilization (%)	Avg SM Utilization (%)	Avg Power (W)	Energy (J)
AthenaPK	1x	563	0.01	7.54	90.09	234.24
	4x	2093	1.78	30.29	88.86	5407.36
BerkeleyGW-Epsilon	1x	30157	2.63	9.04	94.41	319448.05
Cholla-Gravity	1x	615	0.51	13.6	88.43	309.51
	4x	5063	4.45	45.16	138.75	20285.8
Kripke	1x	621	0.27	26.56	123.3	382.24
	4x	5481	3.78	63.21	148.16	12467.54
Cholla-MHD	1x	2175	31.01	72.58	234.24	9849.99
	4x	6753	41.29	88.58	261.64	127249.21
LAMMPS	1x	2321	4.24	63.0	196.79	580.54
	4x	4977	7.13	96.28	258.38	29390.48
WarpX	1x	61453	0.04	33.29	117.14	2588.8
	4x	61453	19.75	77.28	244.32	85756.49

TABLE II: Utilization statistics for selected workflows

For example, while a [throughput×efficiency] product gives equal weight to both metrics, if throughput is more important than efficiency, a [throughput×throughput×efficiency] product can be used to identify workflow co-scheduling configurations that are weighted toward throughput. Often configurations optimized for one metric will be less optimal for the other, necessitating such a product metric to identify the best configuration for given system requirements.

V. Evaluation

For our analysis, we selected 7 representative HPC workloads and tested the effect of various collocation decisions on system throughput and energy. Common to many of the codes we tested is the Kokkos portability library [22], which provides an architecture-agnostic API to adapt HPC application code to various parallel backends, such as OpenMP, CUDA, and HIP. The basic concepts of Kokkos are described in [23]. Even though multiple benchmarks use this library, the utilization characteristics of the individual applications still vary widely. Table II shows the utilization and power profile for each benchmark task for multiple problem sizes. Because many of our benchmarks are physics simulation codes, problem size is usually determined by the number of atoms in the simulation input.

A. Workflow Benchmarks

The benchmarks we selected are drawn from commonly used HPC workloads. We used many exascale benchmarks tested in [24] as well as codes drawn from the NERSC-10 suite [25].

AthenaPK is a flexible framework for astrophysical fluid dynamics simulations available on GitHub [26]. This code combines the hydrodynamics and magnetohydrodynamics solvers of Athena++ [27] with the block-structured adaptive mesh refinement framework of Parthenon [28] and the performance portability of Kokkos. Our benchmark uses three-dimensional hydrolinear wave convergence as a test problem.

BerkeleyGW is a widely-used code in simulation workflows for predicting the optical properties of materials and nanostructure [17]. The Epsilon module computes a material's dielectric function via three main computational kernels. Although we didn't investigate scaling with this benchmark due to resource limitations of our evaluation environment, the computational complexity of the Epsilon module for this test problem increases O(N^4) with the number of atoms in the input.

Cholla (Computational Hydrodynamics on ParaLLel Architectures) is a GPU-native three-dimensional astrophysical hydrodynamics code [29]. Cholla demonstrates nearly ideal scaling to multiple GPUs using MPI. Cholla-MHD adapts the base hydrodynamics code to magnetohydrodynamics (MHD) [30]. For our benchmarks, we selected two test problems: three-dimensional gravitational collapse due to spherical overdensity [31] and three-dimensional Advecting Field Loop [32].

Kripke is a mini-app benchmark produced and maintained by Lawrence Livermore National Laboratory (LLNL) that also serves as a proxy app for the LLNL ARDRA neutral particle transport code [19]. The Kripke code solves the Discrete Ordinance and Diamond Difference discretized steady-state linear Boltzmann equation, but more importantly provides a test environment to investigate how different data layouts affect instruction, thread, and task level parallelism.

LAMMPS is a high-performance molecular dynamics simulation code [16] and is the performance-critical component of Parsplice [33] workflows typically used to simulate defects in energy-relevant materials.

WarpX is an electromagnetic and electrostatic Particle-In-Cell code used in the development of advanced particle accelerators [20]. The test problem used in our evaluation models a beam-driven plasma-wakefield accelerator (PWFA) [34].

B. Workflow Combinations

We evaluated our approach on a number of workflow combinations described in Table III. These combinations feature workloads with a variety of utilization profiles and energy requirements. Figure 2 shows the overall throughput, energy efficiency, and power capping (defined in Section V-C) for both MPS and time-slicing sharing mechanisms. The effectiveness of GPU sharing with MPS for both throughput and energy efficiency varied widely depending on the task compositions of the workflows being co-scheduled. In the combinations we tested, the increase in throughput from GPU sharing using MPS ranged from 0% to 147%. Energy efficiency improvement with MPS similarly ranged from a 2% *decrease* to a 109% increase. It is clear, therefore, that the benefit of co-scheduling workflows on the same set of GPUs depends on

	Workflow 1			Workflow 2			Workflow 3			Workflow 4		
Comb. #	**Benchmark**	**Problem Size**	**# Iter.**	**Benchmark**	**Problem Size**	**# Iter.**	**Benchmark**	**Problem Size**	**# Iter.**	**Benchmark**	**Problem Size**	**# Iter.**
1	AthenaPK	4x	5	LAMMPS	4x	3						
2	Epsilon	1x	1	Athena	8x	1	Athena	4x	14			
3	Kripke	4x	11	WarpX	2x	8						
4	Kripke	4x	13	WarpX	4x	2						
5	Epsilon	1x	1	MHD	4x	2						
6	Gravity	4x	4	Kripke	2x	48						
7	MHD	4x	2	LAMMPS	4x	8						
8	Athena	1x	300	Gravity	1x	50	Athena	1x	300	Gravity	1x	50
9	Athena	1x	300	Gravity	1x	50						
10	MHD	4x	1	LAMMPS	4x	4	MHD	4x	1	LAMMPS	4x	4

TABLE III: Workflow combinations

(1) the resource utilization profile of the workflows and (2) whether energy efficiency or throughput is prioritized as a scheduling consideration.

C. Understanding Impact of Power Capping

Software (SW) power capping occurs when the power draw of a GPU reaches a set maximum threshold. In the case of the NVIDIA A100X GPUs used in our evaluation, that value is 300 watts. The power capping data in Figure 3 shows the relative increase in software power capping for MPS and time-slicing over sequential scheduling for the workflow combinations listed in Table III. GPU power draw depends on a number of factors—such as average GPU clock frequency—but closely correlates with GPU utilization. In other words, the instantaneous power draw will be higher whenever more SMs are active. By definition, SW power capping indicates the SW Power Scaling algorithm is reducing the clock frequency below the requested or default clock frequency [35]. Because clock frequency is reduced, there is potential for throughput to decrease when power capping occurs frequently during workflow execution. Our experiments, however, show that there is no direct correlation between workflow throughput and clock throttling due to SW power capping. Workflow Combination 6 in Figure 3, for example, showed the highest differential in the percentage of execution time where power capping was active for MPS compared to sequential scheduling. At the same time, the throughput for this combination was not significantly lower than other combinations, suggesting that throughput and energy efficiency are influenced more by other factors like contention for GPU compute resources. It is interesting to note that while power capping limits overall power consumption, the resulting increase in task latency from clock throttling seems to cancel out any energy efficiency benefits. A more comprehensive study of the energy effects of power capping (with varying power thresholds) is left to future work.

D. Impact of Cardinality and Scheduling Configuration for Workflow Combinations

In this section we discuss the effect of both cardinality and scheduling configuration on the metrics of interest. Cardinality refers to the number of concurrent workflows scheduled simultaneously, while scheduling configuration more broadly refers to both the number of sequential tasks in each workflow as well as the number of concurrent workflows scheduled.

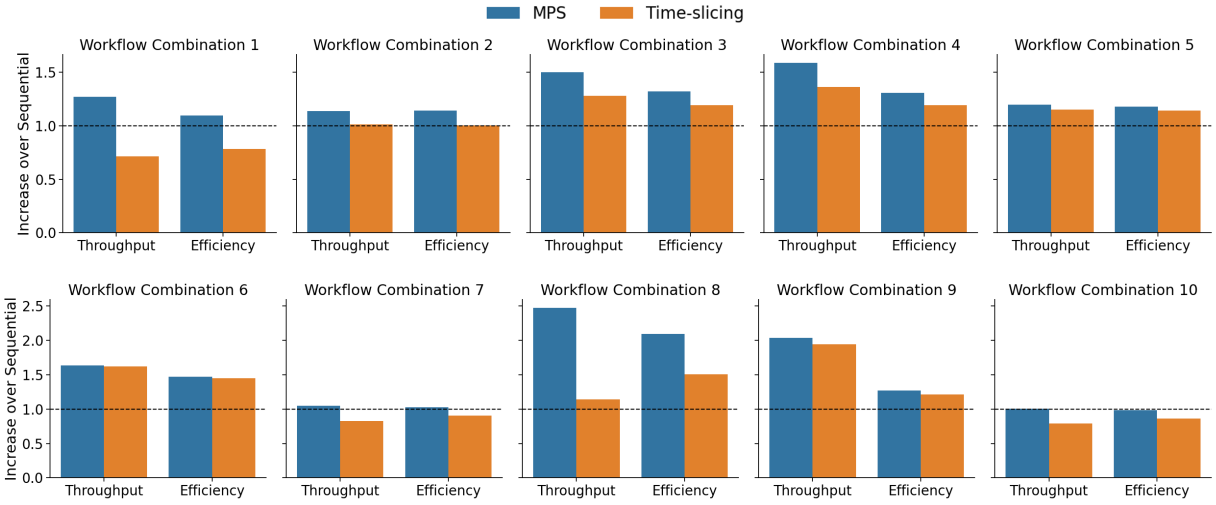


Fig. 2: Throughput and energy efficiency for workflow combinations 1-10

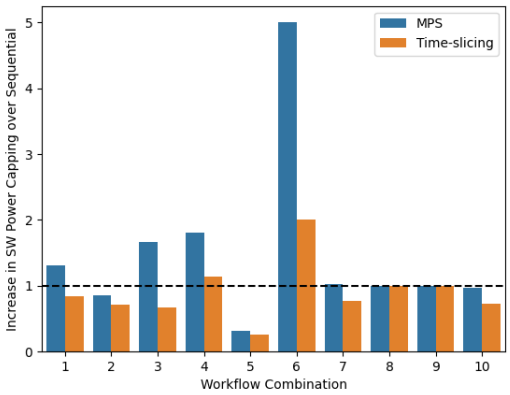


Fig. 3: Percentage of time spent throttling GPU clock frequency due to SW power capping for workflow combinations 1-10

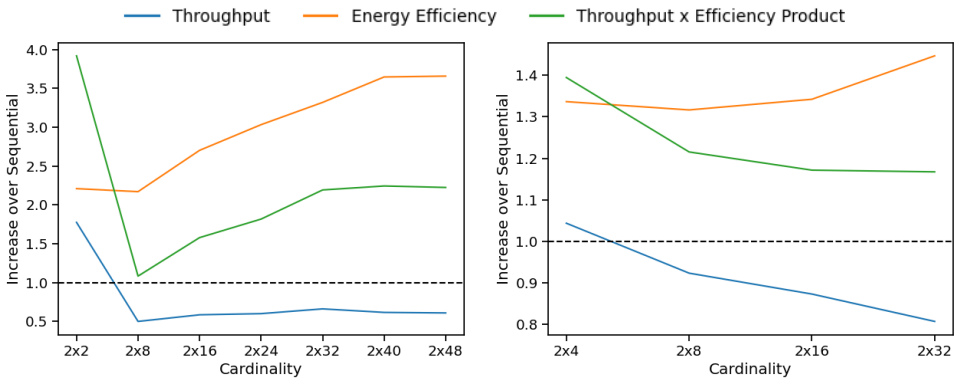
Often workflows consist of multiple independent tasks that can be launched either sequentially or in parallel, which gives freedom in determining the exact scheduling configuration of all tasks (e.g., whether to treat them as single sequential workflows or split them into multiple parallel workflows.

Cardinality. We evaluated the effect of cardinality using workflows comprised of LAMMPS and AthenaPK tasks. LAMMPS is the most resource-intensive workload we tested and AthenaPK is the least. We varied the number of MPS clients while increasing the total number of workloads being scheduled. Figure 4 shows the change in overall throughput and efficiency for the entire set of workflows for each client total over sequential scheduling when using MPS. In the set labels (x-axis), the first number identifies how many sequential tasks were launched for each workflow while the second number refers to the number of workflows launched in parallel. Because MPS outperforms time-slicing in every instance, we omit the time-slicing data here. In these experiments, we ran workflows consisting of multiple instances of the same application to evaluate the effect of scheduling configuration for a high- and low-utilization workflow. For AthenaPK, the low-utilization workflow, as we increased the number of clients launched on the same GPU, throughput dropped considerably from the 2x configuration to the 8x configuration and then changed relatively little as we added additional clients. At the same time, energy efficiency continued to rise as we continued adding clients up to the 48-client maximum. These results are intuitive because throughput drops as the overall utilization and interference increase and energy efficiency increases as we overlap more work. Some key takeaways here are (1) both the high- and low-utilization workflows show similar trends regarding these metrics, (2) the impact of cardinality is greater for the low-utilization workflow, and (3) the throughput benefit of collocation with MPS drops off sharply as we increase cardinality. The product metric in this case simply shows the relative dominance of throughput vs energy efficiency as we add clients. In both Figure 4a and Figure 4b, throughput drops by a greater degree than the increase in energy efficiency, which is reflected in the product of the two. Depending on which metric is more important, workflows can be scheduled using different cardinality. For example, if throughput is selected as the most important measure, the scheduling algorithm should avoid configurations with cardinality greater than 2x2 or 2x4 for these workflows. If energy efficiency is more salient, on the other hand, higher cardinality is more beneficial. The product metric shown in Figure 4 captures this relationship, and depending on the relative priority of throughput and efficiency (or potentially another measure) different product metrics can be used.

Scheduling Configuration. We tested the impact of scheduling configuration again using workflows comprised of LAMMPS and AthenaPK tasks. I this case, the total number of workflow tasks remains constant for each workflow set. Similar to Figure 4, Figure 5 shows the change in overall throughput and energy efficiency for the entire set of workflows for each configuration over sequential scheduling when using MPS. Again, the first number in the configuration label refers to the number of sequential tasks launched by a workflow, and the second refers to the number of workflows launched concurrently. While the results in Figure 4 are intuitive, those in Figure 5 are more surprising. For a workflow consisting of low-utilization tasks like AthenaPK, minimizing the number of concurrent workflows is more beneficial for throughput even if there are enough GPU resources available to schedule more concurrent MPS clients. Holding all other variables equal, scheduling fewer, longer-running workflows yields the most benefit to throughput, while maximizing oversubscription of GPUs yields slightly more benefit to energy efficiency. In the case of AthenaPK, SM utilization is low enough that launching 48 clients does not generate interference. LAMMPS workflows, on the other hand, do not benefit from MPS co-scheduling regardless of the workflow scheduling configuration. The increase in throughput and efficiency peak at around 6% improvement over sequential scheduling, which suggests that workflows involving only LAMMPS tasks are not optimal to be co-scheduled with other LAMMPS tasks. Energy efficiency improvement for MPS scheduling essentially remains constant for both workflows regardless of configuration, which shows that this metric depends on the total number of tasks being scheduled. In general, the more tasks being collocated via MPS sharing, the greater the overall energy efficiency over sequential scheduling.

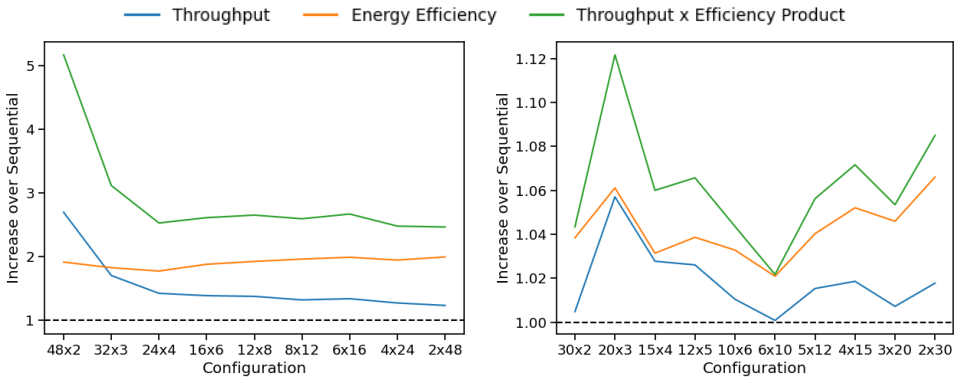
VI. Conclusions and Recommendations

In this work, we evaluated a granularity- and interference-aware scheduling approach that uses MPS to collocate (or co-schedule) HPC workflows on the same set of GPUs. We compiled a set of scientific HPC benchmarks comprised of commonly used simulation codes and full-scale test problems. Using these benchmarks, we evaluated the effect of co-scheduling on both system throughput and energy efficiency. Our evaluation demonstrates both that the granularity of the MPS partition determines the benefit of GPU sharing and that sharing must be interference-aware to maximize benefit.



(a) AthenaPK (b) LAMMPS

Fig. 4: Throughput, energy efficiency, and efficiency-throughput product for AthenaPK and LAMMPS workflow combinations with increasing cardinality (i.e., the number of concurrent workflows scheduled simultaneously)



(a) AthenaPK (b) LAMMPS

Fig. 5: Throughput, energy efficiency, and efficiency-throughput product for different configurations of AthenaPK and LAMMPS workflow combinations

We make the following recommendations for programmers interested in finding optimal scheduling configurations for HPC workflow multi-tenancy with MPS:

1) If throughput is most important, schedule low-utilization workflows in groups of 2-3 and avoid collocating high-utilization workflows together. Needless to say, if the latency of any individual workflow is most important then one should carefully evaluate the cost and benefit of concurrent execution.
2) If energy efficiency is most important, schedule lowest-utilization workflows first and increase cardinality until the decrease in throughput is intolerable.
3) Where possible, pair workflows with opposing power profiles (i.e., low average power with high average power). Though factors like resource contention have more impact on throughput and energy efficiency, minimizing power drawn still boosts GPU performance.

Future Work: This work is limited in scope and can be expanded to build a comprehensive scheduling framework that incorporates the key takeaways from our results. We are currently investigating a model that takes into account different types of GPU interference between workflows—e.g., compute, memory, memory bandwidth—and recommends the best workflow combinations to optimize either throughput or energy efficiency. We intend to incorporate into this model a measure of computational kernel similarity between workflows to minimize offline analysis of all possible combinations. Additionally, we are exploring the applicability of our results to AMD GPU architectures, and investigating other sharing mechanisms in addition to MPS—e.g., Orion and other fine-grained kernel scheduling mechanisms [10], [11].

References

[1] C. Chang, V. L. Deringer, K. S. Katti, V. Van Speybroeck, and C. M. Wolverton, "Simulations in the era of exascale computing," *Nature Reviews Materials*, vol. 8, no. 5, pp. 309–313, May 2023. [Online]. Available: https://doi.org/10.1038/s41578-023-00540-6

[2] A. Vajda, *Multi-core and Many-core Processor Architectures*. Boston, MA: Springer US, 2011, pp. 9–43. [Online]. Available: https://doi.org/10.1007/978-1-4419-9739-5_2

[3] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *Journal of Systems Architecture*, vol. 129, p. 102561, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122001138

[4] D. Milojicic, P. Faraboschi, N. Dube, and D. Roweth, "Future of hpc: Diversifying heterogeneity," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 276–281.

[5] A. Tekin, A. Durak, C. Piechurski, D. Kaliszan, F. A. Sungur, F. Robertsen, and P. Gschwandtner, "State-of-the-art and trends for computing and interconnect network solutions for hpc and ai," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235430195

[6] NVIDIA. (2024) CUDA Toolkit Documentation. [Online]. Available: https://docs.nvidia.com/cuda

[7] CUDA Runtime API Stream Management. [Online]. Available: https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html

[8] NVIDIA. (2024) Multi-Process Service User Guide. [Online]. Available: https://docs.nvidia.com/deploy/mps/#what-mps-is

[9] Multi-Instance GPU User Guide. [Online]. Available: https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html

[10] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 539–558. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/han

[11] F. Strati, X. Ma, and A. Klimovic, "Orion: Interference-aware, fine-grained gpu sharing for ml applications," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1075–1092. [Online]. Available: https://doi.org/10.1145/3627703.3629578

[12] J. Li, G. Michelogiannakis, B. Cook, D. Cooray, and Y. Chen, "Analyzing resource utilization in an hpc system: A case study of nersc's perlmutter," in *High Performance Computing*, A. Bhatele, J. Hammond, M. Baboulin, and C. Kruse, Eds. Cham: Springer Nature Switzerland, 2023, pp. 297–316.

[13] Nersc-10 workload analysis data. [Online]. Available: https://www.nersc.gov/systems/nersc-10/workload-analysis/

[14] Z. Zhao, E. Rrapaj, S. Bhalachandra, B. Austin, H. A. Nam, and N. Wright, "Power analysis of nersc production workloads," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1279–1287. [Online]. Available: https://doi.org/10.1145/3624062.3624200

[15] G. Hautreux and E. Malaboeuf, "Reducing hpc energy footprint for large scale gpu accelerated workloads," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1860–1865. [Online]. Available: https://doi.org/10.1145/3624062.3624268

[16] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[17] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "Berkeleygw: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," *Computer Physics Communications*, vol. 183, no. 6, pp. 1269–1289, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465511003912

[18] W. Jeon, G. Ko, J. Lee, H. Lee, D. Ha, and W. W. Ro, "Chapter six - deep learning with gpus," in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, ser. Advances in Computers, S. Kim and G. C. Deka, Eds. Elsevier, 2021, vol. 122, pp. 167–215. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300905

[19] "Kripke: a simple, scalable, 3d sn deterministic particle transport code," 2019, https://github.com/LLNL/Kripke [Accessed: 07.08.2024].

[20] L. Fedeli, A. Huebl, F. Boillod-Cerneux, T. Clark, K. Gott, C. Hillairet, S. Jaure, A. Leblanc, R. Lehe, A. Myers, C. Piechurski, M. Sato, N. Zaim, W. Zhang, J.-L. Vay, and H. Vincenti, "Pushing the frontier in the design of laser-based electron accelerators with groundbreaking mesh-refined particle-in-cell simulations on exascale-class supercomputers," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–12.

[21] NVIDIA. (2024) Nsight Systems User Guide. [Online]. Available: https://docs.nvidia.com/nsight-systems/UserGuide/index.html

[22] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 805–817, 2022.

[23] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202 – 3216, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001257

[24] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach, T. Evans, M. Ezell, N. Frontiere, A. Georgiadou, J. Glenski, P. Grete, S. Hamilton, J. Holmen, A. Huebl, D. Jacobson, W. Joubert, K. Mcmahon, E. Merzari, S. Moore, A. Myers, S. Nichols, S. Oral, T. Papatheodore, D. Perez, D. M. Rogers, E. Schneider, J.-L. Vay, and P. K. Yeung, "Frontier: Exploring exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607089

[25] National Energy Research Scientific Computing Center. NERSC-10 Benchmark Suite. [Accessed: 12.06.2024]. [Online]. Available: https://gitlab.com/NERSC/N10-benchmarks

[26] Parthenon HPC Lab. Athenapk: a performance portable version of athena++ built on parthenon and kokkos. [Accessed: 07.08.2024]. [Online]. Available: https://github.com/parthenon-hpc-lab/athenapk

[27] J. M. Stone, K. Tomida, C. J. White, and K. G. Felker, "The athena++ adaptive mesh refinement framework: Design and magnetohydrodynamic solvers," *The Astrophysical Journal Supplement Series*, vol. 249, no. 1, p. 4, jun 2020. [Online]. Available: https://dx.doi.org/10.3847/1538-4365/ab929b

[28] P. Grete, J. C. Dolence, J. M. Miller, J. Brown, B. Ryan, A. Gaspar, F. Glines, S. Swaminarayan, J. Lippuner, C. J. Solomon, G. Shipman, C. Junghans, D. Holladay, J. M. Stone, and L. F. Roberts, "Parthenon—a performance portable block-structured adaptive mesh refinement framework," *The International Journal of High Performance Computing Applications*, vol. 37, no. 5, pp. 465–486, 2023. [Online]. Available: https://doi.org/10.1177/10943420221143775

[29] E. E. Schneider and B. E. Robertson, "Cholla: A new massively parallel hydrodynamics code for astrophysical simulation," *The Astrophysical Journal Supplement Series*, vol. 217, no. 2, p. 24, Apr. 2015. [Online]. Available: http://dx.doi.org/10.1088/0067-0049/217/2/24

[30] R. V. Caddy and E. E. Schneider, "Cholla-mhd: An exascale-capable magnetohydrodynamic extension to the cholla astrophysical simulation code," 2024. [Online]. Available: https://arxiv.org/abs/2402.05240

[31] E. E. Schneider and B. E. Robertson. Cholla: A 3d gpu-based hydrodynamics code. [Accessed: 14.05.2024]. [Online]. Available: https://github.com/cholla-hydro/cholla

[32] T. A. Gardiner and J. M. Stone, "An unsplit godunov method for ideal mhd via constrained transport in three dimensions," *Journal of Computational Physics*, vol. 227, no. 8, p. 4123–4141, Apr. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2007.12.017

[33] D. Perez, E. D. Cubuk, A. Waterland, E. Kaxiras, and A. F. Voter, "Long-time dynamics through parallel trajectory splicing," *Journal of Chemical Theory and Computation*, vol. 12, no. 1, pp. 18–28, 2016, pMID: 26605853. [Online]. Available: https://doi.org/10.1021/acs.jctc.5b00916

[34] E. Esarey, P. Sprangle, J. Krall, and A. Ting, "Overview of plasma-based acceleration cocept," *Plasma Science, IEEE Transactions on*, vol. 24, pp. 252 – 288, 05 1996.

[35] NVIDIA. (2024) NVML API Reference Guide. [Online]. Available: https://docs.nvidia.com/deploy/nvml-api/group__nvmlClocksEventReasons.html#group__nvmlClocksEventReasons