ArcHPC

Nexus + Oracle

ARC COMPUTE

Introduction

Controlling advanced accelerated hardware chips, especially NVIDIA GPUs, is critical to remaining an international leader in innovative technology. Even with control, organizations that fail to optimize these underlying hardware systems cannot maintain longer-term competitive advantages. Luckily, the true potential of GPUs remains essentially untapped.

Achieving peak GPU performance has eluded even the most advanced companies due to the current limitations in managing and manipulating how data flows and threads execute on these GPUs. These limitations present a tremendous opportunity that Arc Compute aims to harness with its ArcHPC software suite.

ARC COMPUTE

Table of Contents

Foundational Concepts

- Everything that occurs on a computer relates back to machine code and that code moves around based on read and writes.
- How code executes.
  - Example NVIDIA CUDA Kernel; prominent AI/ML chip
- There are inherent latencies in all architectures between when data is being copied, memory access operations, and when it is being “worked” on, arithmetic operations.
- Code is written in siloed and compartmentalized environments.
  - Team A is building their code to provide X and Team B is building their code to provide Y and neither of them is working together to contemplate whether the hardware it's running on will be fully utilized or optimized.
- GPUs are designed to maximize throughput for parallel computing.
- Code is written as serialized execution so the latency between series of sequential execution is the most important once resource needs are met.

What You’ll Learn

- Underutilized GPUs negatively impact performance; it is impossible to utilize the GPU fully.
- GPU(s) have operational performance triggers, which result in the GPU(s) performing better.
- List of optimization points per stage in the process of running a CUDA Kernel that ArcHPC can manage.
- Execution Operations for kernels to complete applications on GPUs are not rigid but fluid, adaptable and malleable. Achieving complete control over compute and architecture is attainable with low level solutions to dominate control of the accelerated hardware.

Process of Running a CUDA Kernel		
	Stage	Optimization Capabilities of ArcHPC
1	CPU sets up initial states on GPU	N/A
2	CPU uploads system kernels (malloc/free/memcpy/CNP) to the GPU	N/A
3	CPU starts executing the CUDA process	N/A
4	CUDA process creates table on CPU of corresponding kernels to run and name those kernels	N/A
5	CUDA process performs malloc on the GPU side	By default, CUDA malloc is on stream 0, this is a blocking stream. This stream will cause all kernels to stop running until this is finished. By modifying the stream that is represented as stream 0 between different VMs allows unoptimized code to run with corresponding code and experience pseudo syncs on a CPU level. The CPU operates at a faster frequency than the GPU a majority of the time allowing it to finish faster. Longer term we can allow for FPGAs to perform the pseudo sync as well.
6	CUDA process performs memcpy on the GPU side	By default, CUDA memcpy is on stream 0, this is a blocking stream. This stream will cause all kernels to stop running until this is finished. By modifying the stream that is represented as stream 0 between different VMs allows unoptimized code to run with corresponding code and experience pseudo syncs on a CPU level. The CPU operates at a faster frequency than the GPU a majority of the time allowing it to finish faster. Longer term we can allow for FPGAs to perform the pseudo sync as well.
7	CUDA process uploads kernel to execute to GPU side	The kernel upload operation is intercepted and allocated on the CPU for our own use cases, we can also look at the "calling size" of the GPU kernel. This allows us to determine the size of the data without requiring us to analyze the data. We experience latency in this stage for the first initial kernel recording(s) and measurements.
8	CUDA process performs kernel operations	Multiple kernels can be concurrently executed. As a kernel experiences a warp stall(s) other kernels can be completed.
9	CUDA process performs memcpy on the GPU side	By default, CUDA malloc is on stream 0, this is a blocking stream. This stream will cause all kernels to stop running until this is finished. By modifying the stream that is represented as stream 0 between different VMs allows un optimized code to run with corresponding code and experience pseudo syncs on a CPU level. A CPU operates at a faster frequency than the GPUs majority of the time allowing it to finish faster. Longer term we can allow for FPGAs to perform the pseudo sync as well.
10	CUDA process analyzes it on CPU side	N/A

Summary of Functions

Capabilities

- Code Management
- System Management

Abilities

- User Defined Rejections
- Code
    - Machine Code Denial
    - Machine Code Intercept
    - Machine Code Trap
    - Machine Code Replace
    - Machine Code Orchestration
- System
    - System Discovery
    - System Mapping
    - System Spoofing
    - System Compartmentalization

Application Examples & Effects

- Reduce compute time; AI/ML training and inference time reduced by 67.5%
- Increase user/task density 100% to 400% without performance degradation

Performance

- Accelerated AI Training and Inference
- Simulation for advanced personnel training
- Modeling for advanced component and synthetic developments
- Data analytics

Resilience

- Next Generation platforms and programs
- Resistant to Electronic warfare
- Communication systems
- Command and control systems

- Military Grade Code; Cyber warfare counter measures.
- Military Grade Code; Cyber warfare resistant code execution.

RTOS

- Surveillance and Reconnaissance Platforms
- Guided Missile Systems
- Autonomous Vehicles
- Radar Systems

- Task Oriented Power Regulation; savings of 39%+.
- Task Oriented Thermal Regulation.

Cyber Security

- Encryption
- Code Denial
- Machine Code Data Falsification
- Dedicated Computing and Memory Pathways

- GPU Anti Virus.
- Counter Intelligence; instruction spoofing and data contamination.

The applications are not limited to what is shown here as examples. Performance figures can be improved with further development.
Current performance figures represent a beta version benchmarking. Further developments have occurred since.

Introducing

ArcHPC

Case Studies Overview

Overview

Arc Compute conducted four experiments to determine whether increasing the throughput to a GPU could boost its performance. This investigation aimed to activate intrinsic low-level optimization points within the GPU and identify any limitations within our software. ArcHPC Nexus (Beta version) successfully eliminated inherent obstacles found in NVIDIA's fractional GPU solutions, thus enabling all Streaming Multiprocessors (SMs) to be fully accessible for tasks that share GPU resources.

Findings

- Increasing throughput to a GPU and removing barriers that are found in NVIDIA Fractional GPU solutions increases the performance of a GPU while increasing user/task density.
- With Nexus, compute times for workloads, such as AI/ML training, can be reduced by 28.5% to 67.5% without optimizing the code for Nexus.
- The performance of a GPU can be increased by 140% to 308% using the Nexus.
- User/task density can be increased by 100% while increasing performance by 140% to 211%; this results in a 28.5% to 52.6% reduction in compute time.
- Nexus reduces energy consumption for task completion/computations by 13.679% to 38.832%.
- There are many more optimization capabilities to increase performance even further, and they're already being worked on by Arc Compute.

To Discover

Can performance be increased further, and why?

Case Study 1

Performance Benchmarks

The proof of concept was created to explore both the performance enhancements and limitations of ArcHPC Nexus.
Performance benchmarks were conducted for two distinct types of jobs based on their varying L2 cache requirements: a small compute job that progressively decreases in size and a large memory compute job that continues to expand. Despite these changes, the computing operation for each job remained unchanged. The objective was to adjust the L2 cache sizes for these jobs to identify the points at which performance begins to degrade.

This approach aims to enable the classification and consolidation of workloads with different GPU compute demands—specifically, those requiring high compute power versus those with high L2 memory needs—within the same architecture, according to targeted performance levels. The architecture in question is equipped with a 40GB NVIDIA A100.

The three configurations for the tests performed are the following:

- Job-Big performed on full-passthrough (by itself)
- Job-Small performed on full-passthrough (by itself)
- Job-Small and Job-Big performed on a single GPU using ArcHPC Nexus(both started at the same time and ran on the same GPU)

PoC Description

SGEMM runs multiple times (1024 for compute tasks, 1 for memory tasks). (We use cublas) : C A * B + 0.5

Operation:

Start infinite or measure mode — Upload kernels for A, B, and C — Warm up GPU with C = A * B + 0.5 — Start samples — Start timer — Run for N iterations: C = A * B + 0.5 — Stop timer — Take average time over the timer — Stop Samples — Calculate min, max, average and post into a csv file

Case Configurations

Configuration 1: Job-Big — Passthrough

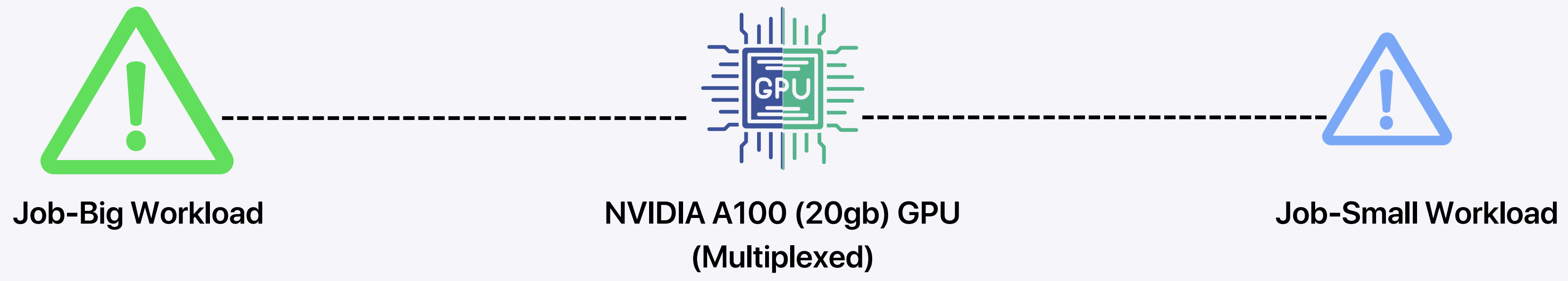
In this configuration, the Job-Big workload is utilizing a full A100 40GB GPU without the use of ArcHPC Nexus.

Configuration 2: Job-Small — Passthrough

In this configuration, the Job-Small workload is utilizing a full A100 40GB GPU without the use of ArcHPC Nexus.

Configuration 2: Job-Big/Job-Small — Shared

Utilizing ArcHPC Nexus, the Job-Big workload and the Job-Small workload are sharing an A100 40GB GPU with both jobs running simultaneously.

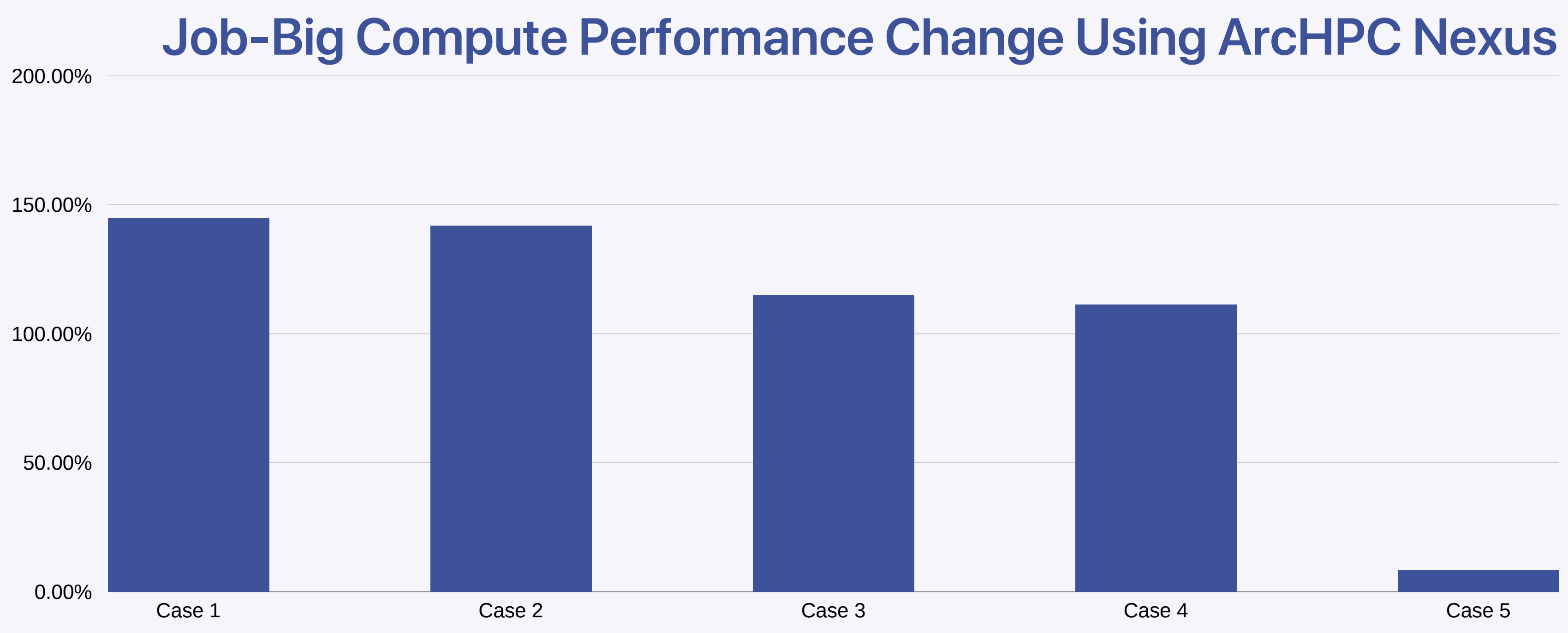


Job-Big Results

Compute time is measured in milliseconds per iteration

Cases	Job-Big Matrices				Size (mb)	L2 Cache Utilization	L2 Cache Total (mb)
Case 0	1024	1024	1024	864	10.75	27%	21.50
Case 1	2048	1024	1024	864	18.13	45%	25.19
Case 2	4096	1024	1024	864	32.88	82%	38.09
Case 3	8192	1024	1024	864	62.38	156%	66.67
Case 4	16364	1024	1024	864	121.23	303%	125.07

Job-Big Summary				
Cases	Idle	Compute	Performance % (+/-)	L2 Cache
Case 0	0.1409	0.1151	144.83%	10.75
Case 1	0.2573	0.2127	141.94%	18.13
Case 2	0.4385	0.4079	115.00%	32.88
Case 3	0.854	0.8079	111.41%	62.38
Case 4	1.6613	3.0672	8.33%	121.23

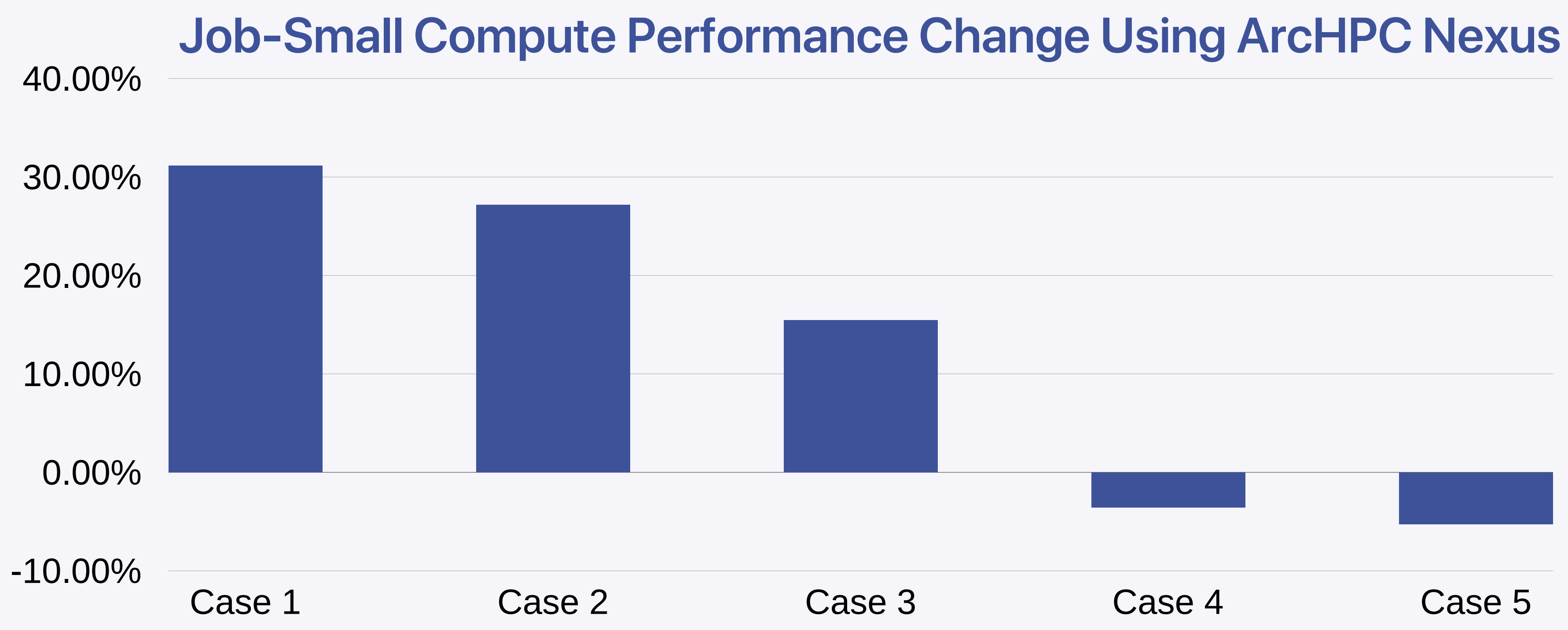


Job-Small Results

Compute time is measured in milliseconds per iteration

Cases	Job-Small Matrices				Size (mb)	L2 Cache Utilization	L2 Cache Total (mb)
Case 0	1024	1024	1024	864	10.75	27%	21.50
Case 1	512	1024	1024	864	7.06	18%	25.19
Case 2	256	1024	1024	864	5.22	13%	38.09
Case 3	128	1024	1024	864	4.30	11%	66.67
Case 4	64	1024	1024	864	3.84	10%	125.07

Job-Small Summary				
Cases	Idle	Compute	Performance % (+/-)	L2 Cache
Case 0	0.1095	0.167	31.14%	10.75
Case 1	0.059	0.0928	27.16%	7.06
Case 2	0.0321	0.0556	15.47%	5.22
Case 3	0.258	0.0535	-3.55%	4.30
Case 4	0.018	0.038	-5.26%	3.84

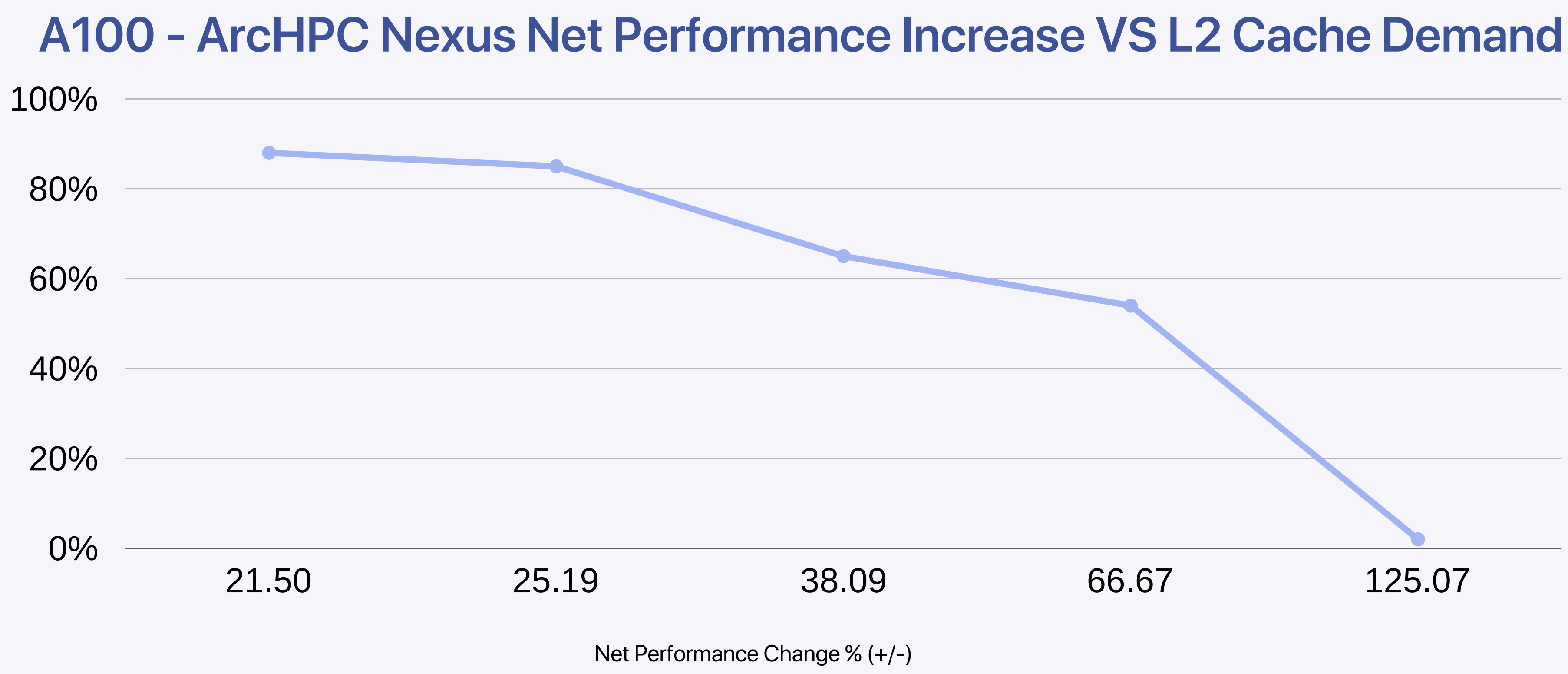


Comparing Results

Compute time is measured in milliseconds per iteration

	Idle Small			Idle Big			Compute Small			Compute Big		
Cases	Ave	Min	Max	Ave	Min	Max	Ave	Min	Max	Ave	Min	Max
Case 0	0.110	0.109	0.137	0.141	0.118	0.158	0.167	0.163	0.167	0.115	0.113	0.119
Case 1	0.059	0.059	0.076	0.257	0.214	0.280	0.093	0.088	0.094	0.213	0.209	0.217
Case 2	0.032	0.032	0.041	0.439	0.413	0.532	0.056	0.050	0.056	0.408	0.407	0.413
Case 3	0.026	0.026	0.033	0.854	0.797	1.042	0.054	0.045	0.054	0.808	0.801	2.297
Case 4	0.018	0.018	0.023	1.661	1.600	2.061	0.038	0.029	0.040	3.067	2.047	4.621

Cases	Small Compute Performance % (+/-)	Big Compute Performance % (+/-)	Net Performance % (+/-)	L2 Cache Total (mb)
Case 0	31.14%	144.83%	88%	21.50
Case 1	27.16%	141.94%	85%	25.19
Case 2	15.47%	115.00%	65%	38.09
Case 3	-3.55%	111.41%	54%	66.67
Case 4	-5.26%	8.33%	2%	125.07



Conclusion

Job-Big: Performance

In Configuration 3, the computation time for the workload limited by memory was 0.1151 milliseconds, in contrast to 0.1409 milliseconds when it used the entire GPU in Configuration 1. Typically, using half the resources in Configuration 1 would result in computation times that are twice as long. However, this did not happen with ArcHPC Nexus. For the workload cases limited by large memory in Configuration 3, Nexus improved performance by between +8% and +144%.

Job-Small: Performance

In Configuration 3, the compute-limited workload finished in 0.167 milliseconds, as opposed to 0.1095 milliseconds when the full GPU was used in Configuration 2. Typically, one might expect Configuration 2 to require double the computation time when provided with half the resources. However, this expectation does not apply to ArcHPC Nexus. For cases of compute-limited workload in Configuration 3, Nexus varied the performance, resulting in changes that ranged from a decrease of 5% to an increase of 31%.

Combined Performance

When considering both small compute-limited and big memory-limited workloads in Configuration 3, we observe a significant performance improvement ranging from +2% to +88% across various cases. This improvement is attributed to the ArcHPC Nexus' capacity to enhance performance and optimize operations for better productive use of GPU resources. Nexus demonstrates the potential for different types of workloads to run on the same GPU, thereby reducing the overall need for hardware by enhancing performance through complete resource utilization.

Reflecting on the outcomes for both the Job-Small and Job-Big, our recommendation is to interleave compute-limited and memory-limited jobs, utilizing up to 62MB of L2 Cache on an NVIDIA A100 GPU, which typically offers 40MB of L2 Cache. This approach can lead to a performance boost of up to 54% while achieving 100% utilization of the GPU. It is important to note that performance sharply declines when exceeding three times the L2 Cache capacity available in the system. Although system specifications and the capabilities of compute chips may vary, it is evident that ArcHPC Nexus facilitates the consolidation of infrastructure by maximizing performance and ensuring complete utilization of GPU resources.

CPU Specifications

CPU op-mode(s)	32-bit, 64-bit
Byte order	Little Endian
Address sizes	46 bits physical, 57 bits virtual
CPU(s)	80
On-lin CPU(s) list	0-79
Thread(s) per core	2
Core(s) per socket	20
Sockets(s)	2
NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Model	106
Model name	Intel(R) Xeon(R) Gold 5320T CPU @ 2.30GHz
Stepping	6
Frequence boost	enabled
CPU MHz	800.209
CPU max MHz	3500
CPU min MHz	800

BogoMIPS	4600.00
Virtualization	VT-x
L1d cache	1.9 Mi B
L1i cache	1.3 Mi B
L2 cache	50 Mi B
L3 cache	60 Mi B
NUMA node0 CPU(s)	0-19,40-59
NUMA node 1 CPU(s)	20-39,60-79
Vulnerability Itlb multihit	Not affected
VulnerabilityL1tf	Not affected
Vulnerability Mds	Not affected
Vulnerability Meltdown	Not affected
Vulnerability Mmio stale data	Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Retbleet	Not affected
Vulnerability Spec store bypass	Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1	Mitigation; usercopy/swapgs barriers and__user pointer sanitization
Vulnerability Spectre v2	Mitigation; enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence
Vulnerability Srbds	Not affected

Case Study 2

Case Study 2

Overview

We increased the sample size repeating the test for Case Study 1, and made improvements to our software's management of the GPU to determine if results would improve or degrade.

Summary

- The results improved further with an increased sample size.
- Increasing throughput to a GPU triggers low-level optimization mechanisms such as memory coalescing, "hot" SMs, and optimal warp scheduling mitigating thread divergence.
- NVIDIA's CuBLAS library also presents opportunities for further scheduling of work. This can be observed in the FFMA (Fused Multiply-Add) warp stall. To understand this better, compare the latencies involved in moving data within the A100 architecture with the time required to complete an arithmetic operation.
- Significant HBM2 to L2 and SM occupancy waste occurs where ArcHPC Nexus is not present to optimize compute resources. The lack of optimization and compute resource waste is present throughout cases for "Idle Big" and "Idle Small" when compared to cases where ArcHPC Nexus was optimizing the completion of tasks "Compute Big" and "Compute Small" concurrently. This shows that HBM2 to L2 and SM occupancy cannot be used to determine compute requirement for tasks; granular pipeline metrics are more reliable key indicators of resource demands - see table for "Small Kernel" and "Large Kernel" in conjunction to warp stall table, Chart of SM Utilization for Matrices Experiment and performance results of tests.

To Discover

- Can performance be increased further?
- Is this applicable across multiple GPUs?
- What is the implication if a GPU doesn't have as much work?
- What are the situations that cause an SM to power down?

Job-Small & Big Results

Compute time is measured in milliseconds per iteration

Cases	Job-Small				Size (mb)
Case 0	1024	1024	1024	864	10.75
Case 1	512	1024	1024	864	7.06
Case 2	256	1024	1024	864	5.21
Case 3	128	1024	1024	864	4.29
Case 4	64	1024	1024	864	3.83
Case 5	64	1024	1024	864	3.83
Case 6	64	1024	1024	864	3.83
Case 7	64	1024	1024	864	3.83
Case 8	64	1024	1024	864	3.83
Case 9	64	1024	1024	864	3.83

Cases	Job-Big				Size (mb)
Case 0	1024	1024	1024	864	10.75
Case 1	2048	1024	1024	864	18.12
Case 2	4096	1024	1024	864	32.87
Case 3	8192	1024	1024	864	62.37
Case 4	16364	1024	1024	864	121.23
Case 5	32728	1024	1024	864	239
Case 6	65456	1024	1024	864	475
Case 7	130912	1024	1024	864	946
Case 8	261824	1024	1024	864	1889
Case 9	523648	1024	1024	864	3775

Job-Small & Big Summary

Compute time is measured in milliseconds per iteration

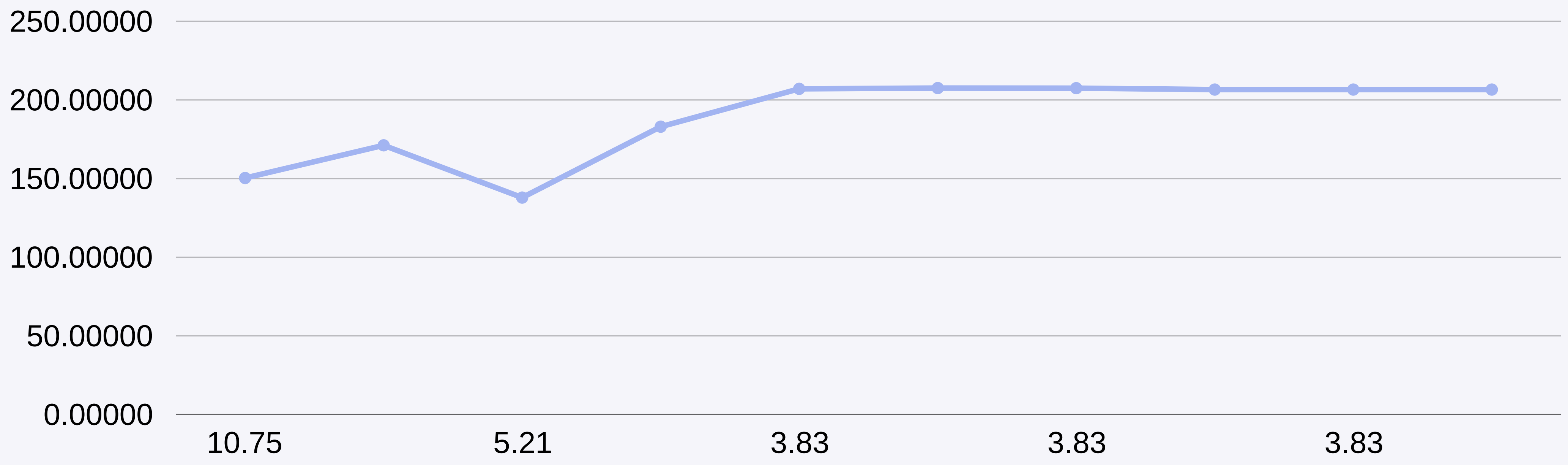
Job-Small Summary					
Cases	Idle	Compute	Performance % (+/-)	L2 Cache	L2 Cache Utilization
Case 0	0.1422	0.1136	150.3521127%	10.75	27%
Case 1	0.0888	0.0655	171.1450382%	7.06	18%
Case 2	0.0571	0.048	137.9166667%	5.21	13%
Case 3	0.0474	0.0335	182.9850746%	4.29	11%
Case 4	0.0347	0.0226	207.0796460%	3.83	10%
Case 5	0.0346	0.0225	207.5555556%	3.83	10%
Case 6	0.0349	0.0227	207.4889868%	3.83	10%
Case 7	0.0348	0.0227	206.6079295%	3.83	10%
Case 8	0.0348	0.0227	206.6079295%	3.83	10%
Case 9	0.0348	0.0227	206.6079295%	3.83	10%

Job-Big Summary					
Cases	Idle	Compute	Performance % (+/-)	L2 Cache	L2 Cache Utilization
Case 0	0.1532	0.1138	169.2442882%	10.75	27%
Case 1	0.2664	0.212	151.3207547%	18.12	45%
Case 2	0.4731	0.4078	132.0255027%	32.87	82%
Case 3	0.8538	0.8069	111.6247366%	62.37	156%
Case 4	1.6677	2.0455	64.0603764%	121.23	303%
Case 5	3.2037	4.403	45.5235067%	239	598%
Case 6	6.3065	8.8414	42.6584025%	475	1187%
Case 7	12.4931	17.6131	41.8614554%	946	2366%
Case 8	24.9244	35.4783	40.5050411%	1889	4723%
Case 9	49.6432	71.0849	39.6729826%	3775	9437%

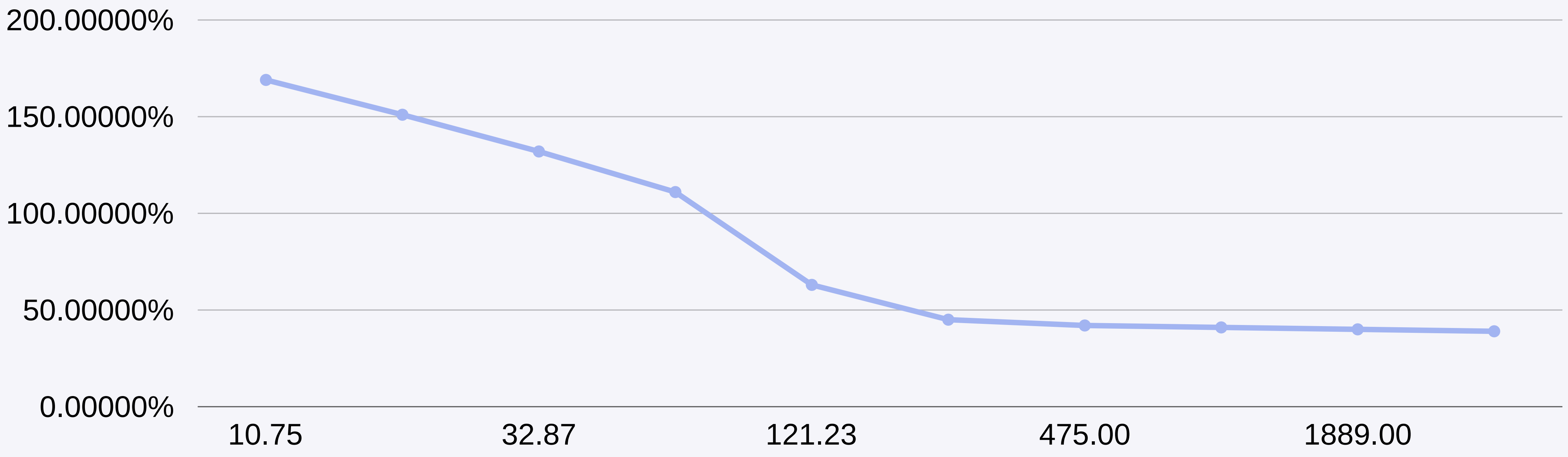
Job-Big & Small

Performance VS L2 Cache

Job Small/Compute Limited Performance vs L2 Cache (Case 0 to Case 9)



Job Big/Memory Limited Performance vs L2 Cache (Case 0 to Case 9)



Job-Big & Small
Performance VS L2 Cache

Chart of SM Utilization for Matrices Experiment - Case by Case

Job-small Summary				
	Idle Big/Jb pass (Memory Limited)	Idle Small/Js pass (Compute Limited)	Compute Big/Jb time when shared (Memory Limited)	Compute Small/Js time when shared (Compute Limited)
Case 0	SM Utils	SM Utils	SM Utils	SM Utils
	82%	82%	54%	44%
	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth
	8%	8%	2%	1%
Case 1	SM Utils	SM Utils	SM Utils	SM Utils
	89%	73%	58%	40%
	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth
	8%	7%	2%	2%
Case 2	SM Utils	SM Utils	SM Utils	SM Utils
	93%	70%	71%	27%
	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth
	13%	7%	6%	2%
Case 3	SM Utils	SM Utils	SM Utils	SM Utils
	96%	60%	82%	16%
	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth
	46%	6%	29%	6%
Case 4	SM Utils	SM Utils	SM Utils	SM Utils
	97%	50%	90%	8%
	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth
	85%	6%	60%	5%
Case 5	SM Utils	SM Utils	SM Utils	SM Utils
	98%	48%	76%	22%
	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth	HBM2 to L2 Bandwidth
	15%	6%	12%	3%

Warp Stall Deadtimes Between Thread Execution and Cycles Elapsed During Memory Access Operations

Source	ampere_sgemm_128x32_nn	ampere_sgemm_64x32_sliced1x4_nn	splitKreduce_kernel
ISETP.NE.AND	95	504	57
	0.238%	21.799%	10.459%
STS	6829	602	0
	17.082%	26.038%	0.000%
FFMA	28948	660	100
	72.412%	28.547%	18.349%
MOV	150	125	125
	0.375%	5.407%	22.936%
STG.E.EF.STRONG.GPU	2	76	0
	0.005%	3.287%	0.000%
LDS	2564	68	0
	6.414%	2.941%	0.000%
BRA	112	40	4
	0.280%	1.730%	0.734%
IADD3	138	73	7
	0.345%	3.157%	1.284%
N/A	100	54	0
	0.250%	2.336%	0.000%
LEA	121	29	1
	0.303%	1.254%	0.183%
IMAD.WIDE	402	15	5
	1.006%	0.649%	0.917%
FADD	0	9	231
	0.000%	0.389%	42.385%
MEMBAR.GPU	2	4	0
	0.005%	0.173%	0.000%
FMUL	6	3	0
	0.015%	0.130%	0.000%
LDG	268	0	7
	0.670%	0.000%	1.284%
LOP3	171	27	0
	0.428%	1.168%	0.000%
P2R	14	6	0
	0.035%	0.260%	0.000%
PLOP3	4	2	0
	0.010%	0.087%	0.000%
SHF.R.U32	10	5	0
	0.025%	0.216%	0.000%
BAR.SYNC.DEFER_BLOCKING	31	2	0
	0.078%	0.087%	0.000%
BMSK	5	0	0
	0.013%	0.000%	0.000%
SHF.L	1	2	0
	0.003%	0.087%	0.000%
ULDC	0	0	3
	0.000%	0.000%	0.550%
HFMA2.MMA	0	0	1
	0.000%	0.000%	0.183%
Total %	99.990%	99.740%	99.266%
Total WSSNI	39977	2312	545

% Warp Stall(Not Issued)
81% to 100%
61% to 80%
41% to 60%
21% to 40%
0% to 20%

Memory Type	CPI (cycles)
Global memory	290
L2 cache	200
L1 cache	33
Shared Memory (ld/st)	(23/19)

Small Kernel vs Large Kernel

Case #	Small Kernel	Large Kernel
0	ampere_sgemm_128x32_nn	ampere_sgemm_128x32_nn
1	ampere_sgemm_128x32_nn	ampere_sgemm_128x32_nn
2	ampere_sgemm_64x32_sliced1x4_nn	ampere_sgemm_128x32_nn
3	ampere_sgemm_64x32_sliced1x4_nn	ampere_sgemm_128x32_nn
4	ampere_sgemm_64x32_sliced1x4_nn + splitKreduce_kernel	ampere_sgemm_128x32_nn
5	ampere_sgemm_64x32_sliced1x4_nn + splitKreduce_kernel	ampere_sgemm_128x32_nn
6	ampere_sgemm_64x32_sliced1x4_nn + splitKreduce_kernel	ampere_sgemm_128x32_nn
7	ampere_sgemm_64x32_sliced1x4_nn + splitKreduce_kernel	ampere_sgemm_128x32_nn
8	ampere_sgemm_64x32_sliced1x4_nn + splitKreduce_kernel	ampere_sgemm_128x32_nn
9	ampere_sgemm_64x32_sliced1x4_nn + splitKreduce_kernel	ampere_sgemm_128x32_nn

Instructions Clock Cycles for the (Ampere A100) GPU

TABLE V
INSTRUCTIONS CLOCK CYCLES FOR THE (*Amepere A100*) GPU

PTX	SASS	cycles	PTX	SASS	cycles
Add / sub instruction			**Min/Max instructions**		
add.u16	UIADD3	2	Min.u16	ULOP3.LUT+UISETP.LT.U32.AND+USEL	8
addc.u32	IADD3.X	2	min.u32	IMNMX.U32	2
add.u32	IADD	2	min.u64	UISETP.LT.U32.AND+2*USEL	8
add.u64	UIADD3.x+ UIADD3	4	min.s16	PRMT+IMNMX	4
add.s64	UIADD3.x+UIADD3	4	min.s32	IMNMX	2
add.f16	HADD	2	Min.s64	UISETP.LT.U32.AND+UISETP.LT.AND.EX+2*USEL	8
add.f32	FADD	2	min.f16	HMNMX2+PRMT	4
add.f64	DADD	4	min.f32	FMNMX	2
Mul instruction			min.f364	DSETP.MIN.AND+IMAD.MOV.U32+UMOV+FSEL	10
mul.wide.u16	LOP3.LUT+IMAD	4	**Neg instruction**		
mul.wide.u32	IMAD	4	neg.s16	UIADD3+UPRMT	5
mul.lo.u16	LOP3.LUT+IMAD	4	neg.s32	IADD3	2
mul.lo.u32	IMAD	2	neg.s64	IMAD.MOV.U32+HFMA2.MMA+MOV+UIADD3	10
mul.lo.u64	IMAD	2	neg.f32	FADD or IMAD.MOV.U32 *	2
mul24.lo.u32	PRMT + IMAD	3	neg.f64	DADD+(UMOV)	4
mul24.hi.u32	UPRMT+USHF.R.U32.HI+IMAD.U32+PRMT	9	**FMA instruction**		
mul.rn.f16	HMUL2	2	fma.rn.f16	HFMA2	2
mul.rn.f32	FMUL	2	fma.rn.f32	FFMA	2
mul.rn.f64	DMUL	4	fma.rn.f64	DFMA	4
MAD Instruction			**Sqrt Instruction**		
mad.lo.u16	LOP3.LUT+IMAD	4	sqrt.rn.f32	[multiple instrs including MUFU.RSQ]	190-235
mad.lo.u32	FFMA	2	sqrt.approx.f32	[multiple instrs including MUFU.SQRT]	2-18
mad.lo.u64	IMAD	2	sqrt.rn.f64	[multiple insts including MUFU.RSQ64]	260-340
mad24.lo.u32	SGXT.U32 + IMAD	4	**Rsqrt Instruction**		
mad24.hi.u32	USHF.R.U32.HI+UIMAD.WIDE.U32+2*UPRMT+IADD3	11	rsqrt.approx.f32	[multiple insts including MUFU.RSQ]	2-18
mad.rn.f32	FFMA	2	rsqrt.approx.f64	MUFU.RSQ64H	8-11
mad.rn.f64	DFMA	4	**Rcp Instruction**		
Sad Instruction			rcp.rn.f32	[multiple insts including MUFU.RCP]	198
sad.u16/s16	(2*LOP3) +ULOP3+ VABSDIFF	6	rcp.approx.f32	[multiple insts including MUFU.RCP]	23
sad.u32/s32	VABSDIFF +IMAD (1 IMAD + 1 Umov for 3 instrs)	3	rcp.rn.f64	[multiple insts including MUFU.RCP64H]	244
sad.u64/s64	UISETP.GE.U32.AND+UIADD+IADD	10	ex2.approx.f32	FSTEP + FMUL + MUFU.EX2 + FMUL	14
Div / Rem Instruction			**Pop Instruction**		
rem/div.u16/s16	multiple instructions	290	popc.b32S	POPC	6
rem/div.s32/u32	multiple instructions	66	popc.b64	2*UPOPC + UIADD3	7
rem/div.u64/s64	multiple instructions	420	**Clz Instruction**		
div.rn.f32	multiple instructions	525	clz.b32	FLO.U32 + IADD	7
div.rn.f64	multiple instructions	426	clz.b64	UISETP.NE.U32.AND+USEL+UFLO.U32+2*UIADD3	13
Abs Instruction			**Bfind Instruction**		
abs.s16	PRMT+IABS+PRMT	4	bfind.u32	FLO.U32	6
abs.s32	IABS	2	bfind.u64	FLO.U32+ISETP.NE.U32.AND+IADD3+BRA	164
abs.s64	UISETP.LT.AND+UIADD3.X +UIADD3+2*USEL	11	bfind.s32	FLO	6
abs.f16	PRMT	1	bfind.s64	multiple instructions	195
abs.ftz.f32	FADD.FTZ	2	**testp Instruction**		
abs.f64	DADD or (DADD+UMOV)	4	testp.normal.f32	IMAD.MOV.U32+2*ISETP.GE.U32.AND	0 or 6
Brev Instruction			testp.subnor.f32	ISETP.LT.U32.AND	0 or 6
brev.b32	BREV + SGXT.U32	2	testp.normal.f64	2*UISETP.LE.U32.AND+2*UISETP.GE.U32.AND	13
brev.b64	2*UBREV+MOV	6	testp.subnor.f64	UISETP.LT.U32.AND+2*UISETP.GE.U32.AND.EX	8
copysign Instruction			**Other Instruction**		
copysign.f32	2*LOP3.LUT or 1.5*LOP3.LUT	4	sin.approx.f32	FMUL + MUFU.SIN	8
copysign.f64	2*ULOP3.LUT+IMAD.U32+*MOV	6	cos.approx.f32	FMUL.RZ+MUFU.COS	8
and/or/xor Instruction			lg2.approx.f32	FSETP.GEU.AND+FMUL+MUFU.LG2+FADD	18
and.b16	LOP3.LUT or 1.5*LOP3.LUT	2	ex2.approx.f32	FSETP.GEU.AND+2*FMUL+MUFU.EX2	18
and.b32	LOP3.LUT	2	ex2.approx.f16	MUFU.EX2.F16	6
and.b64	ULOP3.LUT	2-3	tanh.approx.f32	MUFU.TANH	6
Not Instruction			tanh.approx.f16	MUFU.TANH.F16	6
not.b16	LOP3.LUT	2	bar.warp.sync;	NOP	changes
not.b32	LOP3.LUT	2	fns.b32	multiple instructions	79
not.b64	2*ULOP3.LUT	4	cvt.rzi.s32.f32	F2I.TRUNC.NTZ	6
lop3 Instruction			setp.ne.s32	ISETP.NE.AND	10
lop3.b32	IMAD.MOV.U32+LOP3.LUT	4	mov.u32 clock	CS2R.32	2
cnot Instruction			**Bfi Instruction**		
cnot.b16	ULOP3.LUT+ISETP.EQ.U32.AND+SEL	5	bfi.b32	3*PRMT+2*IMAD.MOV+SHF.L.U32+BMSK+LOP3.LUT	11
cnot.b32	UISETP.EQ.U32.AND+USEL	4	bfi.b64	UMOV+USHF.L.U32+(UIADD3+ULOP3.LUT)*	5
cnot.b64	multiple instructions	11	**dp4a.u32/s32 Instruction**		
bfe Instruction			dp4a.u32.u32	IMAD.MOV.U32+IDP.4A.U8.U8	135-170
bfe.s32/.u32	3*PRMT+2*IMAD.MOV+SHF.R.U32.HI+SGXT/.U32	11	**dp2a.u32/s32 Instruction**		
bfe.u64	UMOV+USHF.L.U32+(UIADD3+ULOP3.LUT)*	5	dp2a.lo.u32.u32	IMAD.MOV.U32+IDP.2A.LO.U16.U8	135-170
bfe.s64	multiple instructions	14			

Case Study 3

Case Study 3

Overview

We ran multiple tests to determine if the performance increase found in Case Study 1 is capable across multiple GPUs; this will determine if this solution applies to tasks that require multiple GPUs. We also tested the implications of insufficient “work” for a GPU.

Summary

- ArcHPC Nexus can increase performance across multiple GPUs.
- Some tasks are memory-bound and others are compute-bound which affects their completion time.
- Being compute-bound and memory-bound is relative.
- Having less work for a GPU negatively impacts its performance in completing an arithmetic operation.
- Fine-tuning task environment and or inflight kernel modification is crucial to keeping performance in the compute "Goldilocks Zone"; this is where performance is greater than 100% for fractionalized accelerated hardware, GPUs.
- Working through various task deployment topologies mimicking transitionary superpositioning of tasks as they are completed, stopped, and started in a real-world environment shows performance remains in the compute "Goldilocks Zone". ArcHPC Nexus performance and utilization benefits remain consistent.

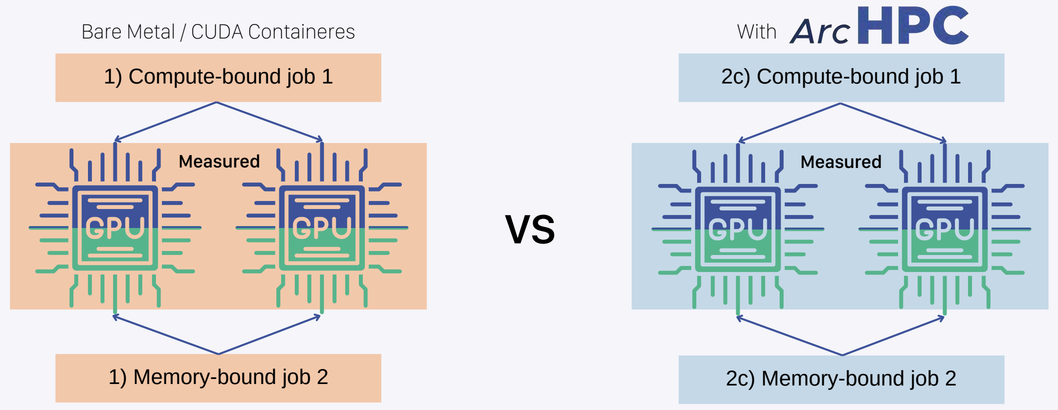
To Discover

- Can performance be increased even further?
- How does this translate to a real-world application?

Case Study 3: 1

	Compute Matricies	Memory Matricies
Case 0	1024	1024
Case 1	512	2048
Case 2	256	4096
Case 3	128	8192
Case 4	64	16384

Same jobs are running simultaneously in both scenarios

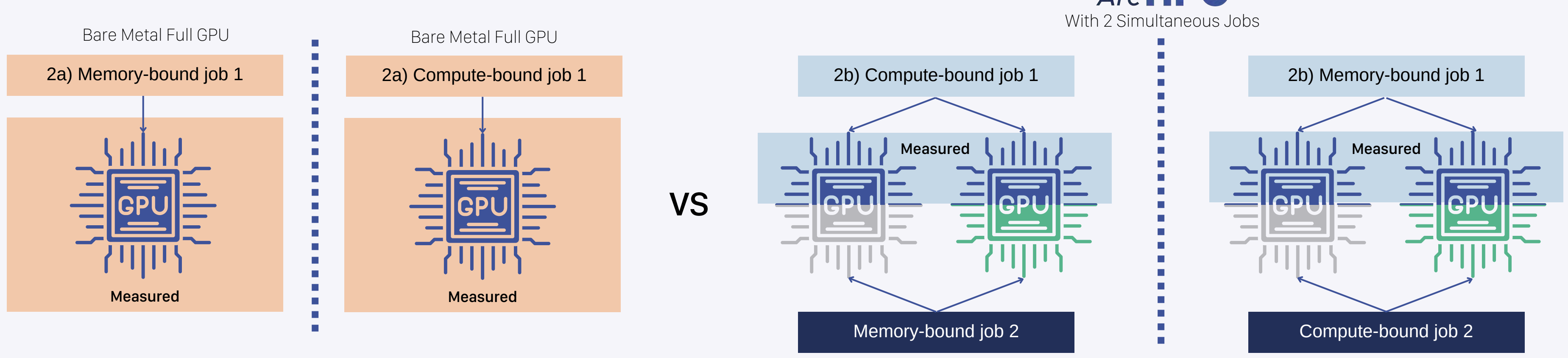


	1) Compute shared double - 0ms - Baremetal (CUDA Containers)			2c) Compute shared double - 0ms - ARC HPC			1) Memory shared double - 0ms - Baremetal (CUDA Containers)			2c) Memory shared double - 0ms - ARC HPC		
	Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (0ms)			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (0ms)			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (0ms)			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (0ms)		
	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
Case 0	0.2657	0.2583	0.2679	0.0581	0.0579	0.0588	0.2656	0.2583	0.2689	0.0581	0.0579	0.0587
Case 1	0.2892	0.2742	0.295	0.0336	0.0334	0.0355	0.2928	0.2912	0.2965	0.1085	0.1083	0.1098
Case 2	0.3786	0.3399	0.3865	0.0248	0.0245	0.0279	0.3863	0.3854	0.3886	0.209	0.2088	0.2097
Case 3	0.5592	0.504	0.5824	0.0191	0.0184	0.0205	0.5824	0.5813	0.5844	0.4115	0.4112	0.4123
Case 4	0.9087	0.6543	0.9921	0.0122	0.0115	0.0145	0.9936	0.9918	0.9972	1.0507	1.0497	1.0516
				Avg Performance Change		Time saved				Avg Performance Change		Time saved
				457%		0.2076				457%		0.2075
				861%		0.2556				270%		0.1843
				1527%		0.3538				185%		0.1773
				2928%		0.5401				142%		0.1709
				7448%		0.8965				95%		-0.0571

This test proves that ARC HPC better manages resources. ROI per case (operating margin factored in) is listed and reflects the performance increase on datacenter investments. ARC HPC cases (except Case 4 memory bound comparisons) substantially surpassed performance of Baremetal CUDA Containers. Comparing the results from "1) Memory shared double - 0ms - Baremetal (CUDA Containers)", "2a) Memory Limited - Baremetal full GPU" and "2c) Memory Shared double - 0ms ARC HPC" we can conclude that memory limitation has caused a performance degradation in Case 4.

Case Study 3: 2b

Two identical jobs are running simultaneously with job 2 using less resources



	2a) Compute limited - Baremetal full GPU			2b) Compute shared single - ARC HPC			2a) Memory Limited - Baremetal full GPU			2b) Memory shared single - ARC HPC		
	Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 - 20GB}			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 - 20GB}		
	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
Case 0	0.1233	0.1211	0.1469	0.098	0.0967	0.1187	0.1227	0.1212	0.1571	0.0983	0.0968	0.1191
Case 1	0.074	0.072	0.0924	0.0644	0.0628	0.079	0.2234	0.2217	0.2583	0.1591	0.1558	0.1834
Case 2	0.0482	0.0465	0.0585	0.0441	0.0427	0.0537	0.4231	0.4218	0.4811	0.2699	0.269	0.2912
Case 3	0.0439	0.0422	0.0528	0.0419	0.0404	0.0518	0.8414	0.8347	0.8906	0.4831	0.4781	0.5143
Case 4	0.0308	0.0285	0.0381	0.0293	0.0275	0.0349	1.728	1.7127	1.7412	1.0026	0.9945	1.0203
				Avg Performance Change		Time saved				Avg Performance Change		Time saved
				126%		0.0253				125%		0.0244
				115%		0.0096				140%		0.0643
				109%		0.0041				157%		0.1532
				105%		0.002				174%		0.3583
				105%		0.0015				172%		0.7254

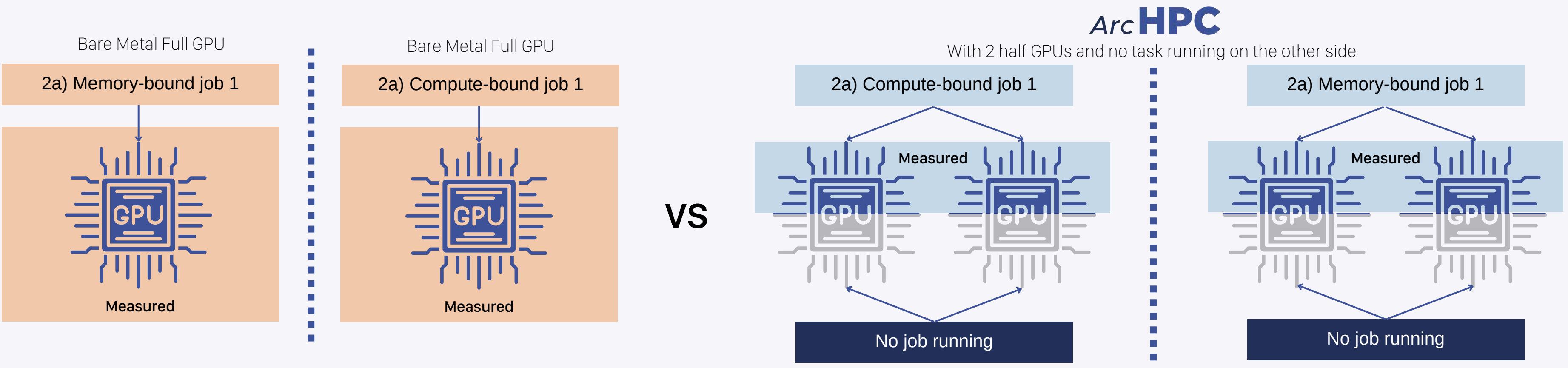
This test proves that ARC HPC will capture available resources that are idle with fewer available resouces that are idle/under utilized. ROI per case (operating margin factored in) is listed and reflects the performance increase on datacenter investments and accomplishing two tasks at the same time. "Compute shared single" decreasing in performance highlights that smaller jobs can benefit from seeing fewer compute resources meaning working with ARC HPC you could increase the density by limiting the size of the VM based on the size of the task.

	Compute Matricies	Memory Matricies
Case 0	1024	1024
Case 1	512	2048
Case 2	256	4096
Case 3	128	8192
Case 4	64	16384

Case Study 3: 2a

	Compute Matricies	Memory Matricies
Case 0	1024	1024
Case 1	512	2048
Case 2	256	4096
Case 3	128	8192
Case 4	64	16384

Two identical jobs, one utilizing a whole GPU, the other utilizing two half GPUs simultaneously

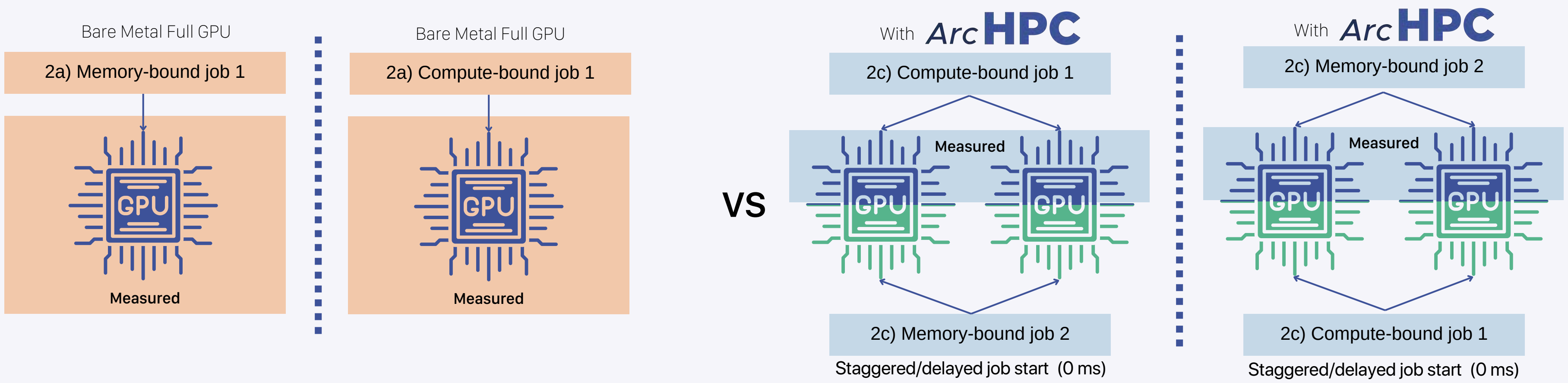


	2a) Compute limited - Baremetal full GPU			2a) Compute limited - ARC HPC			2a) Memory Limited - Baremetal full GPU			2a) Memory limited - ARC HPC		
	Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) N/A - not running not allocated - 1 NVIDIA A100 40 GB {#1 and #2} (No allocation)			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) N/A - not running not allocated - 1 NVIDIA A100 40 GB {#1 and #2} (No allocation)		
	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
Case 0	0.1233	0.1211	0.1469	0.0619	0.0608	0.0784	0.1227	0.1212	0.1571	0.0625	0.0609	0.0793
Case 1	0.074	0.072	0.0924	0.0374	0.0361	0.0465	0.2234	0.2217	0.2583	0.1123	0.1112	0.1402
Case 2	0.0482	0.0465	0.0585	0.0243	0.0236	0.0289	0.4231	0.4218	0.4811	0.2122	0.2115	0.2517
Case 3	0.0439	0.0422	0.0528	0.0227	0.0216	0.0277	0.8414	0.8347	0.8906	0.4204	0.4149	0.4636
Case 4	0.0308	0.0285	0.0381	0.0157	0.0148	0.0196	1.728	1.7127	1.7412	0.8631	0.8535	0.8916
				Avg Performance Change		Time saved				Avg Performance Change		Time saved
				199%		0.0614				196%		0.0602
				198%		0.0366				199%		0.1111
				198%		0.0239				199%		0.2109
				193%		0.0212				200%		0.421
				196%		0.0151				200%		0.8649

This test proves that ARC HPC will capture available resources that are idle. ROI per case (operating margin factored in) is listed and reflects the performance increase on datacenter investments. Since all cases and test surpass full pass through Meta could halve infrastructure seeing a significant savings on their bottom line. Compute limited workload is decreasing in performance because workloads that are smaller benefit from having access to fewer resources with ARC HPC can mitigate.

Case Study 3: 2c-1

Two identical jobs are running simultaneously



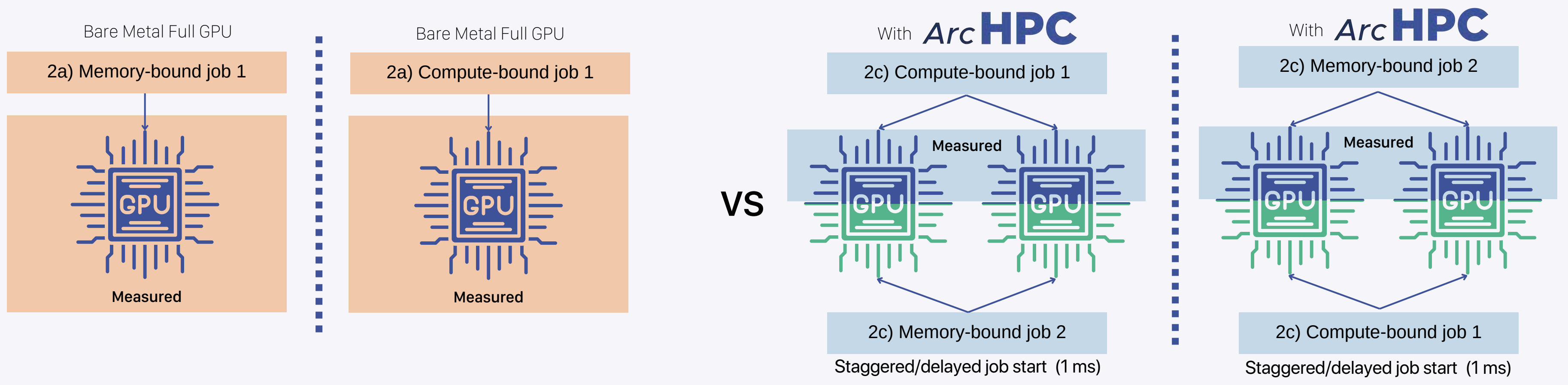
	2a) Compute limited - Baremetal full GPU			2c) Compute shared double - 0ms - ARC HPC			2a) Memory Limited - Baremetal full GPU			2c) Memory shared double - 0ms - ARC HPC		
	Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (0ms)			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (0ms)		
	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
Case 0	0.1233	0.1211	0.1469	0.0581	0.0579	0.0588	0.1227	0.1212	0.1571	0.0581	0.0579	0.0587
Case 1	0.074	0.072	0.0924	0.0336	0.0334	0.0355	0.2234	0.2217	0.2583	0.1085	0.1083	0.1098
Case 2	0.0482	0.0465	0.0585	0.0248	0.0245	0.0279	0.4231	0.4218	0.4811	0.209	0.2088	0.2097
Case 3	0.0439	0.0422	0.0528	0.0191	0.0184	0.0205	0.8414	0.8347	0.8906	0.4115	0.4112	0.4123
Case 4	0.0308	0.0285	0.0381	0.0122	0.0115	0.0145	1.728	1.7127	1.7412	1.0507	1.0497	1.0516
				Avg Performance Change		Time saved				Avg Performance Change		Time saved
				212%		0.0652				211%		0.0646
				220%		0.0404				206%		0.1149
				194%		0.0234				202%		0.2141
				230%		0.0248				204%		0.4299
				252%		0.0186				164%		0.6773

ROI per case (operating margin factored in) is listed and reflects the performance increase on datacenter investments and accomplishing two tasks at the same time with 0 delays between both tasks starting on the GPU and between iterations. By minimizing task delays ARC HPC kept the SMs "hot" 2c) Compute Shares 0ms was able to push through the inherit over provisioning SM core assignment degradation. The merits of remaining "hot" (simulating production enviornments) increased performance to the highest amount for compute across all tests; also seen in memory bound and degrading as we reach limits of hardware memory saturation and pipeline.

	Compute Matricies	Memory Matricies
Case 0	1024	1024
Case 1	512	2048
Case 2	256	4096
Case 3	128	8192
Case 4	64	16384

Case Study 3: 2c-2

Two identical jobs are running simultaneously



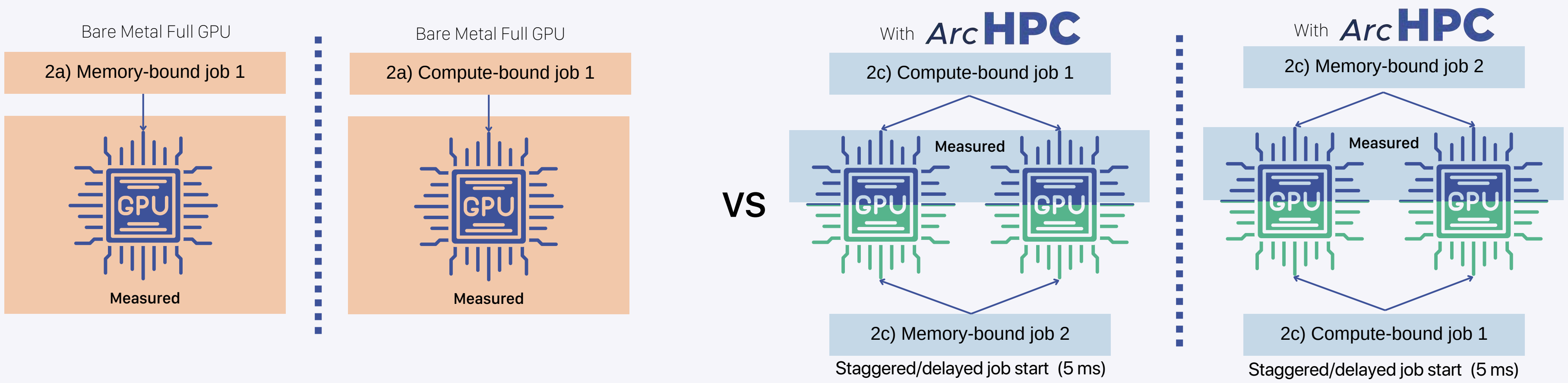
	2a) Compute limited - Baremetal full GPU			2c) Compute shared double - 1ms - ARC HPC			2a) Memory Limited - Baremetal full GPU			2c) Memory shared double - 1ms - ARC HPC		
	Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (1ms)			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (1ms)		
	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
Case 0	0.1233	0.1211	0.1469	0.0637	0.0605	0.0821	0.1227	0.1212	0.1571	0.0629	0.0606	0.0845
Case 1	0.074	0.072	0.0924	0.0392	0.0368	0.045	0.2234	0.2217	0.2583	0.1125	0.1108	0.1447
Case 2	0.0482	0.0465	0.0585	0.0275	0.0236	0.0395	0.4231	0.4218	0.4811	0.2113	0.2106	0.2359
Case 3	0.0439	0.0422	0.0528	0.0287	0.0218	0.043	0.8414	0.8347	0.8906	0.4125	0.4117	0.4472
Case 4	0.0308	0.0285	0.0381	0.0298	0.0155	0.0445	1.728	1.7127	1.7412	1.0524	1.0486	1.1034
				Avg Performance Change		Time saved				Avg Performance Change		Time saved
				194%		0.0596				195%		0.0598
				189%		0.0348				199%		0.1109
				175%		0.0207				200%		0.2118
				153%		0.0152				204%		0.4289
				103%		0.001				164%		0.6756

ROI per case (operating margin factored in) is listed and reflects the performance increase on datacenter investments and accomplishing two tasks at the same time with 1ms delays between start time of every iteration of task 2 on the GPU. We believe the dergradation of 2c) Compute shared double - 1ms - ARC HPC versus 2c) Compute shared double - 0ms - ARC HPC is a result of a loss in "hot" SM core optimization and a bad cycle time and Cache Coherence or L2 Cache efficiency overriding GPU Memory scheduling. The theory behind bad cycle time is due to 2c) Compute shared double - 5ms - ARC HPC completing faster than 2c) Compute shared double - 1ms.

	Compute Matricies	Memory Matricies
Case 0	1024	1024
Case 1	512	2048
Case 2	256	4096
Case 3	128	8192
Case 4	64	16384

Case Study 3: 2c-3

Two identical jobs are running simultaneously



	2a) Compute limited - Baremetal full GPU			2c) Compute shared double - 5ms - ARC HPC			2a) Memory Limited - Baremetal full GPU			2c) Memory shared double - 5ms - ARC HPC		
	Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (5ms)			Task/workload/job (1) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems Fill entire GPU - 1 x NVIDIA A100 40GB			Task/workload/job (2) Time to complete task(s) 'n' iterations measured in seconds N = 1024 Run through circular buffer of SGEMM problems – NVIDIA BLAS problems VM (1) fill half of each GPU {#1 and #2 - 20GB on each GPU} VM (2) fill half of each GPU {#1 and #2 - 20GB on each GPU} - Staggered/delayed job starts (5ms)		
	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
Case 0	0.1233	0.1211	0.1469	0.0619	0.0608	0.0801	0.1227	0.1212	0.1571	0.0618	0.0607	0.0783
Case 1	0.074	0.072	0.0924	0.0379	0.0367	0.048	0.2234	0.2217	0.2583	0.1121	0.1111	0.1503
Case 2	0.0482	0.0465	0.0585	0.0252	0.0237	0.0304	0.4231	0.4218	0.4811	0.2119	0.2111	0.2475
Case 3	0.0439	0.0422	0.0528	0.024	0.021	0.0294	0.8414	0.8347	0.8906	0.4179	0.4119	0.4457
Case 4	0.0308	0.0285	0.0381	0.0178	0.0147	0.0232	1.728	1.7127	1.7412	0.9126	0.8504	0.9526
				Avg Performance Change		Time saved				Avg Performance Change		Time saved
				199%		0.0614				199%		0.0609
				195%		0.0361				199%		0.1113
				191%		0.023				200%		0.2112
				183%		0.0199				201%		0.4235
				173%		0.013				189%		0.8154

ROI per case (operating margin factored in) is listed and reflects the performance increase on datacenter investments and accomplishing two tasks at the same time with 5ms delays between start time of every iteration of task 2 on the GPU.

	Compute Matricies	Memory Matricies
Case 0	1024	1024
Case 1	512	2048
Case 2	256	4096
Case 3	128	8192
Case 4	64	16384

Case Study 4

Discovery

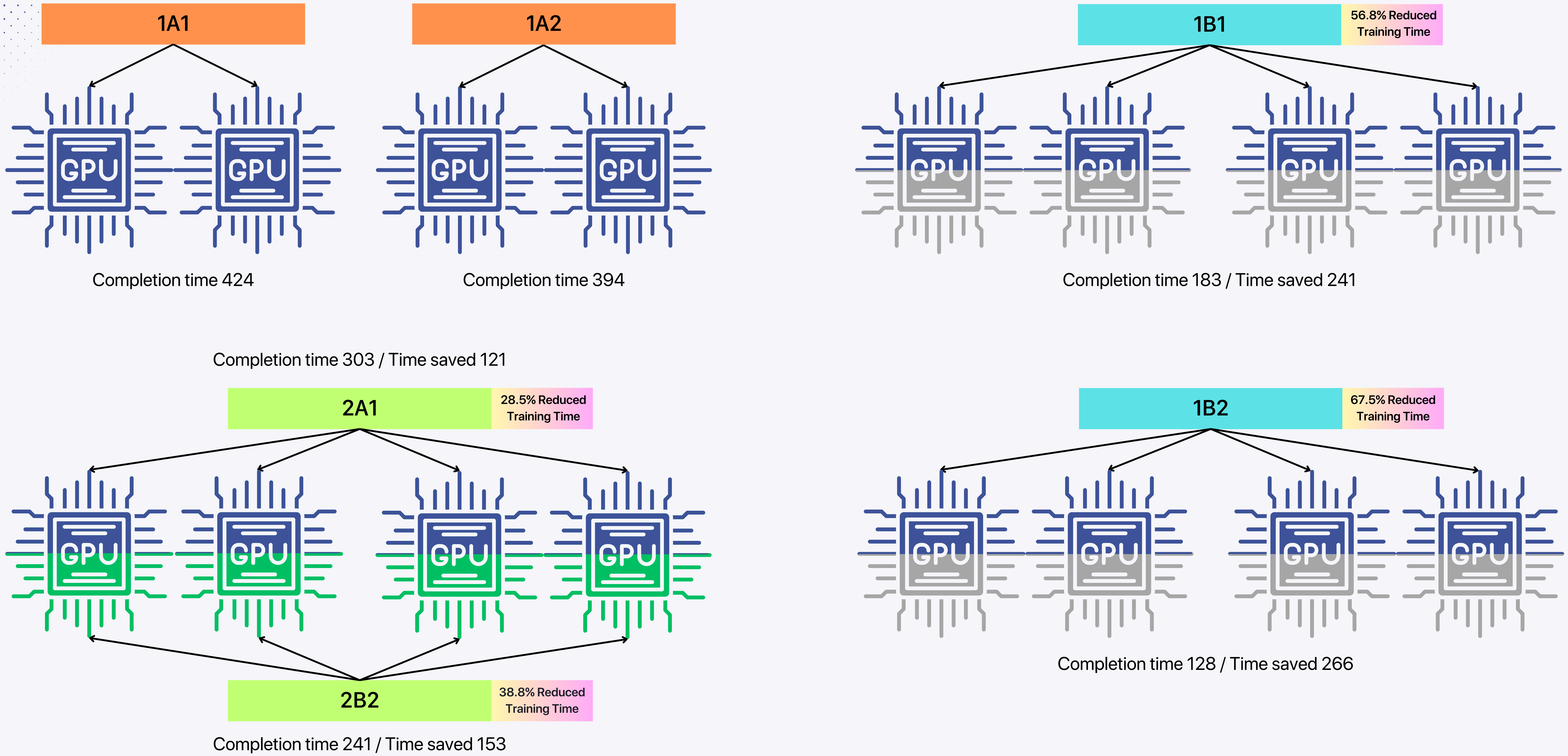
Arc Compute engaged in discussions with the director of AI/HPC infrastructure of a leading AI/ML company, focusing on the challenges of improving utilization within their HPC environment. Low GPU utilization was a primary issue within the company's AI infrastructure, and proper job scheduling utilizing SLURM wasn't a sufficient fix. Having failed to resolve this problem up to this point, Arc Compute piqued the director's interest with a software solution that could drastically improve VRAM allotment and SM utilization.

Impact

The proposition was met with enthusiasm from the AI/ML company, as Arc Compute demonstrated how ArcHPC Nexus could significantly accelerate LLM training and inference times. Furthermore, this solution offers improvements in performance per watt, contributing to a considerable decrease in both the carbon footprint and overall energy usage from a supply, operational, and scaling perspectives. Impressively, these advantages are obtainable without necessitating any optimizations of the company's code for Nexus.

Proposal

To counter these challenges, Arc Compute proposed the implementation of ArcHPC Nexus, a tailored solution aimed at boosting GPU performance. Nexus enhances thread execution per clock cycle, optimizes task compute environments, and increases user/task density. Nexus uniquely facilitates the concurrent running of two tasks, thereby enabling additional arithmetic operations during memory access operations of other tasks. This method of task execution not only optimizes GPU throughput but also ensures tasks are processed together, diverging from the company's existing architecture of isolated compute environments. This leads to quicker task completion times while simultaneously reducing the need for extensive infrastructure.


1A1
Completion time 424
1A2
Completion time 394
1B1
56.8% Reduced Training Time
Completion time 183 / Time saved 241
Completion time 303 / Time saved 121
2A1
28.5% Reduced Training Time
1B2
67.5% Reduced Training Time
Completion time 128 / Time saved 266
2B2
38.8% Reduced Training Time
Completion time 241 / Time saved 153
GPU

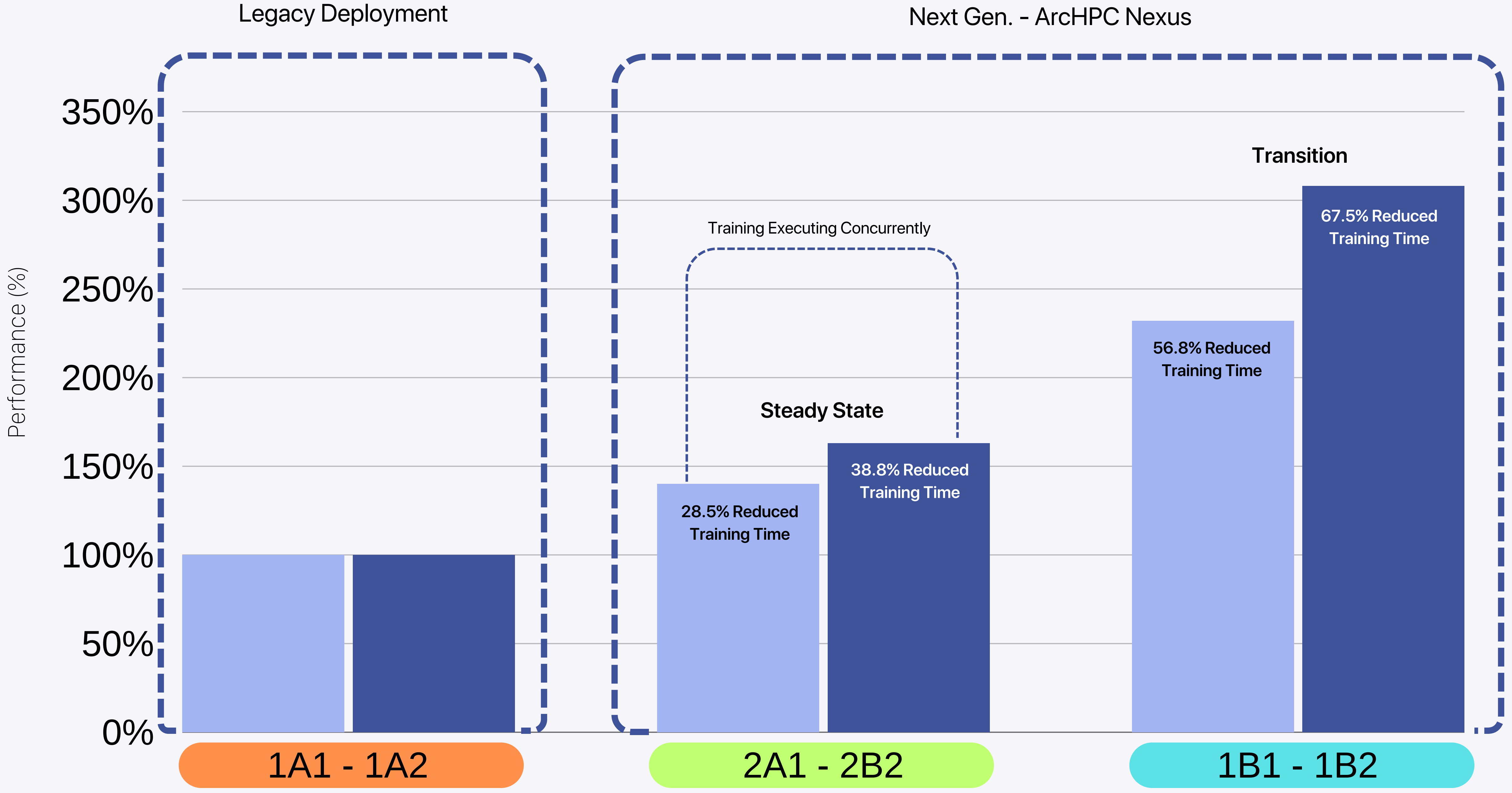
Results

- Increase GPU performance 1.4x to 3x
- Increased user/task density by 100%; reduced infrastructure need by 50%
- Reduce energy expenditure between 13.679% to 38.832%
- Reduce compute time between 29% to 67.5%

Training of LLAMA Model

https://github.com/OpenAccess-AI-Collective/axolotl

No changes to code. Trained as is. Variation are solely configurations of hardware



Training of LLAMA Model

https://github.com/OpenAccess-AI-Collective/axolotl

No changes to code. Trained as is. Variation are solely configurations of hardware

1A1	1A2	2A1	2B2	1B1	1B2
No ArcHPC Nexus	No ArcHPC Nexus	ArcHPC Nexus	ArcHPC Nexus	ArcHPC Nexus	ArcHPC Nexus
BFLOAT16	FP16	BFLOAT16	FP16	BFLOAT16	FP16
Batch Size 16	Batch Size 32	Batch Size 16	Batch Size 32	Batch Size 16	Batch Size 32
2 GPUs	2 GPUs	4 GPUs	4 GPUs	4 GPUs	4 GPUs
A100 40gb SXM	A100 40gb SXM	A100 40gb SXM	A100 40gb SXM	A100 40gb SXM	A100 40gb SXM
Full Pass Through	Full Pass Through	Half per GPU (20gb each GPU) 4 X 20gb Same VRAM as 2 full GPUs	Half per GPU (20gb each GPU) 4 X 20gb Same VRAM as 2 full GPUs	Half per GPU (20gb each GPU) 4 X 20gb Same VRAM as 2 full GPUs	Half per GPU (20gb each GPU) 4 X 20gb Same VRAM as 2 full GPUs
Sole Task Running	Sole Task Running	2B2 Running on other half Two tasks running performing double the work as 1A1 code not optimized for ArcHPC	2B2 Running on other half Two tasks running performing double the work as 1A1 code not optimized for ArcHPC	Sole task running Code not optimized for ArcHPC Nexus	Sole task running Code not optimized for ArcHPC Nexus
Completion Time (Minutes)	Completion Time (Minutes)	Completion Time (Minutes)	Completion Time (Minutes)	Completion Time (Minutes)	Completion Time (Minutes)
424	394	303	241	183	128
N/A	N/A	Compared to 1A1	Compared to 1A2	Compared to 1A1	Compared to 1A2
		Time Saved	Time Saved	Time Saved	Time Saved
		121	153	241	266
		Performance	Performance	Performance	Performance
		140%	163%	232%	308%
		ROI per ($) spent	ROI per ($) spent	ROI per ($) spent	ROI per ($) spent
		2.798679868	3.269709544	2.316939891	3.078125
kWh Usage	kWh Usage	kWh Usage	kWh Usage	kWh Usage	kWh Usage
5.65	5.25	4.04	3.21	4.88	3.41
($) Savings Per kWh	($) Savings Per kWh	($) Savings per kWh	($) Savings per kWh	($) Savings per kWh	($) Savings per kWh
N/A	N/A	28.538%	38.832%	13.679%	35.025%

Why ArcHPC?

a. **Intercepts kernel data**
    i. Required to know the type of binary which is being run on the GPU.
    ii. Allows for the creation of antivirus for GPUs.
b. **Custom GPU selector APIs**
    i. Allows for sysadmins to create their APIs for how to select a GPU on a cluster.
    ii. Predefined selectors allow for information to be added to justify thermal limits or power draw limits on the edge/green datacenters.
    iii. Databinning rules allow for datacenter/cluster administrators to ensure allocatable rules are not deleted.
c. **Custom Kernel Extraction API**
    i. Extracting a CUDA kernel can be sent through a user-defined kernel extraction API.
    ii. Allowing for the preemption of kernel modules based on the VM's priority.
    iii. Can provide mechanisms for restructuring the kernel into several components to run.
d. **Custom in VM plugins**
    i. Users can create their plugins for the use of updating/providing inter-VM communications.
e. **Custom GPU Communication API**
    i. Provide a custom mechanism for communication to and from the GPU on each task (encryption/compression)

Problems

Industry Problem

- Scarcity of GPU resources
- Underutilized GPU investments
- Long compute times when working with large amounts of data
- Increasing energy demand to power compute environments
- Hardware limitations struggling to keep up with software demand
- Mix and match GPU products to meet the demand

Impact of Problem

- Slower product rollouts and software advancement
- Difficulty keeping pace with larger entities that command more GPU allotments from vendors
- Displeasure among employees who have work impacted due to limited computing resources
- Difficulty justifying additional HPC investments while current resources are under-utilized

Summary of Root Problem

- Even the most optimized code has latencies during memory access operations
- Missed opportunities to execute additional arithmetic operations during memory access operations impact GPUs negatively
- Current GPU management solutions cause slowdowns when revealing all compute resources to tasks running concurrently on the same hardware, and are limited to splitting tasks/users across single GPUs
- Current solutions that can split single GPUs for concurrent task execution across the entire resource are difficult to use and do not innately work with prominent job schedulers
- Current GPU management solutions cannot granularly administer compute resources perfectly to calibrate and tune the most optimal compute environment for instruction execution

Defacto Industry Solutions

DEFACTO SOLUTIONS	PROS	CONS
Job schedulers	• Widely available • Easy to use	• Cannot address root problem • Can degrade performance • Cannot granularly manage compute environments • Cannot set or prioritize performance for business objectives
Manual task matching	• Addresses root problem • Increases performance of accelerated hardware • Full control of code optimization cycle	• Reliant on ability to acquire human capital capable of low-level coding and translating between various hardware architectures • Not scalable • Time intensive process • Limited to human capabilities • Cannot address changes to business objectives or operations on the fly • Bureaucratic red tape to execute • Limits production code update potential for product managers and software developers • Process must be restarted for broad updates • Security posture only as strong as the weakest task

Limitation of Other Solutions

	NVAIE/VGPU	MPS	JIT Linking	Fractional timesliced GPUs	MIG
Only works on server architectures	X	X	X	X	X
Does not provide kernel use data so cannot use to determine drain on the GPUs	X	X	X	X	X
Cannot predict what the current power draw/thermal increase	X	X	X	X	X
Cannot preempt lower priority kernels	X	X	X	X	X
No selection of the best GPU to use	X	X	X	X	X
Does not fix the null stream problem	X			X	X
Only time-sliced solutions		X			
Cannot integrate with common job schedulers like SLURM		X			
Requires both CUDA programs to be compiled with a specific flag			X		
Cannot increase/decrease to accommodate workload requiring higher capabilities on the engineerings side					X

The *Arc* HPC Effect

Nexus

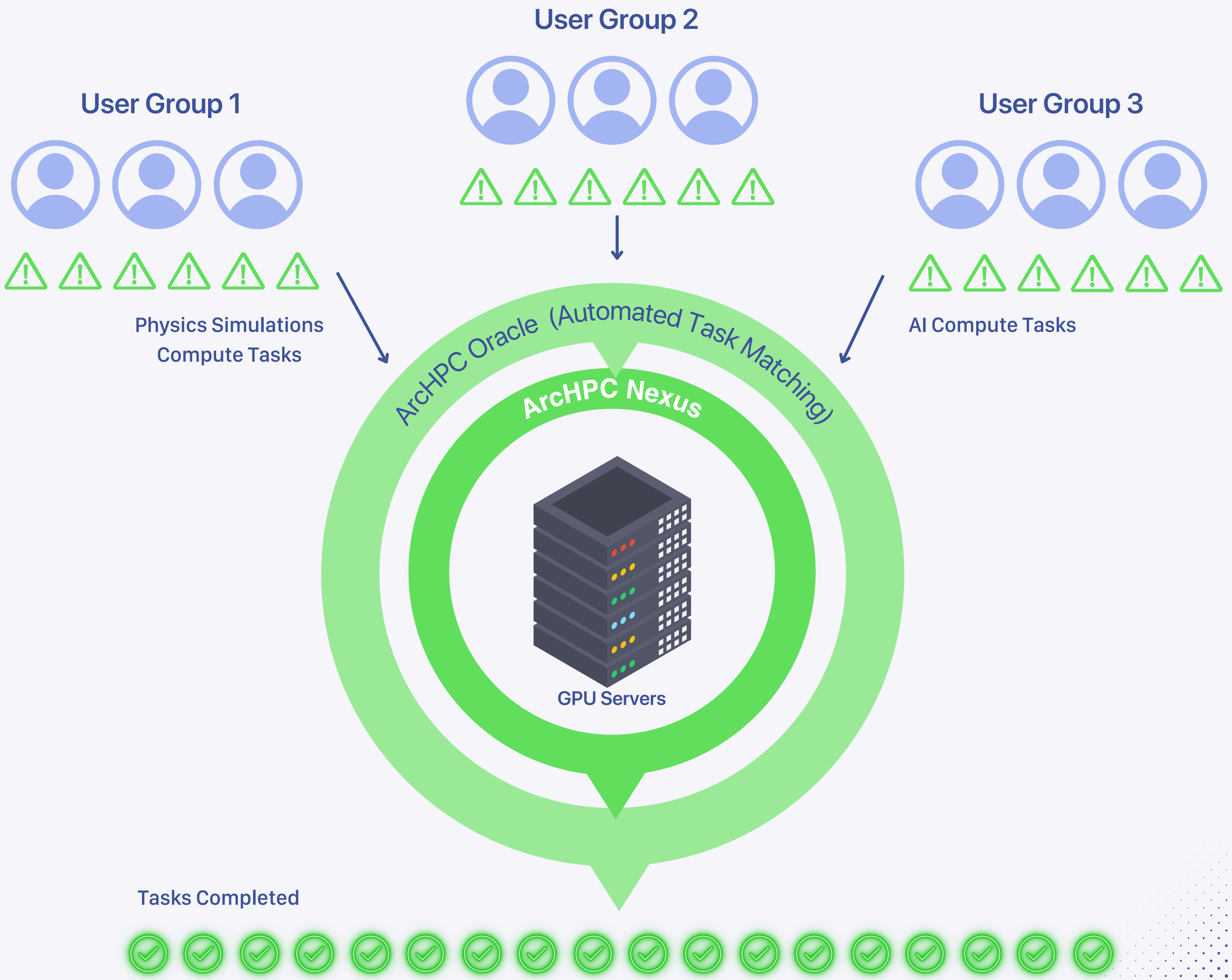
- Creates the environment to maximize utilization and performance
- Manages the HPC environments
- Increases throughput enabling users to increase user/task density
- Manages multiple accelerator types simultaneously

Oracle

- Automates task matching and task deployment
- Manages low-level operational execution of instructions in the HPC environment
- Increases accelerated hardware performance through enterprise scalable control

ArcHPC Suite

up to	up to	as low as
100%	5X	1/6
Utilization of GPU Resources	Boost to GPU Performance	Hardware Requirements



The ArcHPC Method

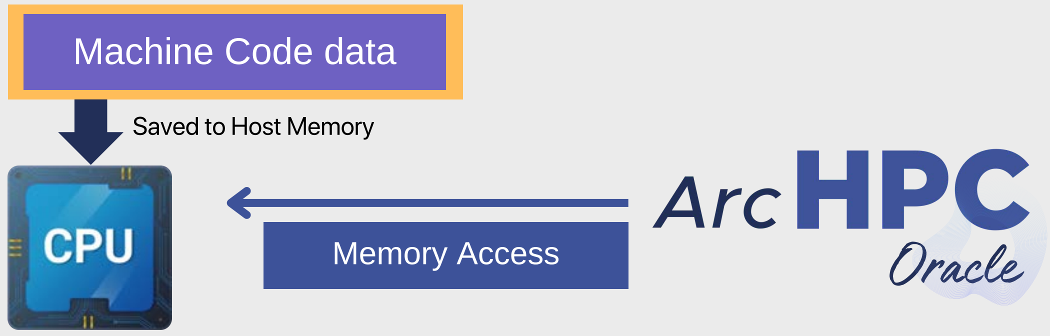
1

ArcHPC Nexus intercepts machine code at the load stage dedicated to the accelerated hardware (NVIDIA, AMD, etc.).

2

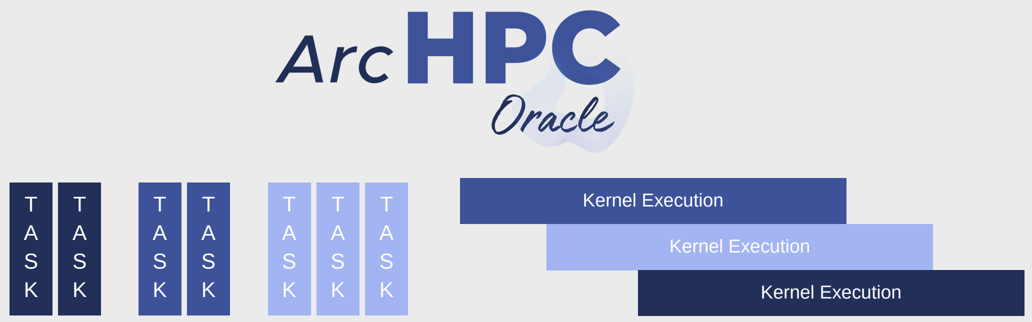
ArcHPC Nexus becomes aware of the instructions for execution and the data size, but not the data contents.

3



The source code is saved to the host memory, independent of ArcHPC Nexus's operation. ArcHPC Oracle accesses the host memory (CPU).

4



ArcHPC Oracle can pair, match tasks, and stagger kernel execution times using the information intercepted from machine codes.

Following the ArcHPC method, a performance increase is seen even for HBM-bound tasks.

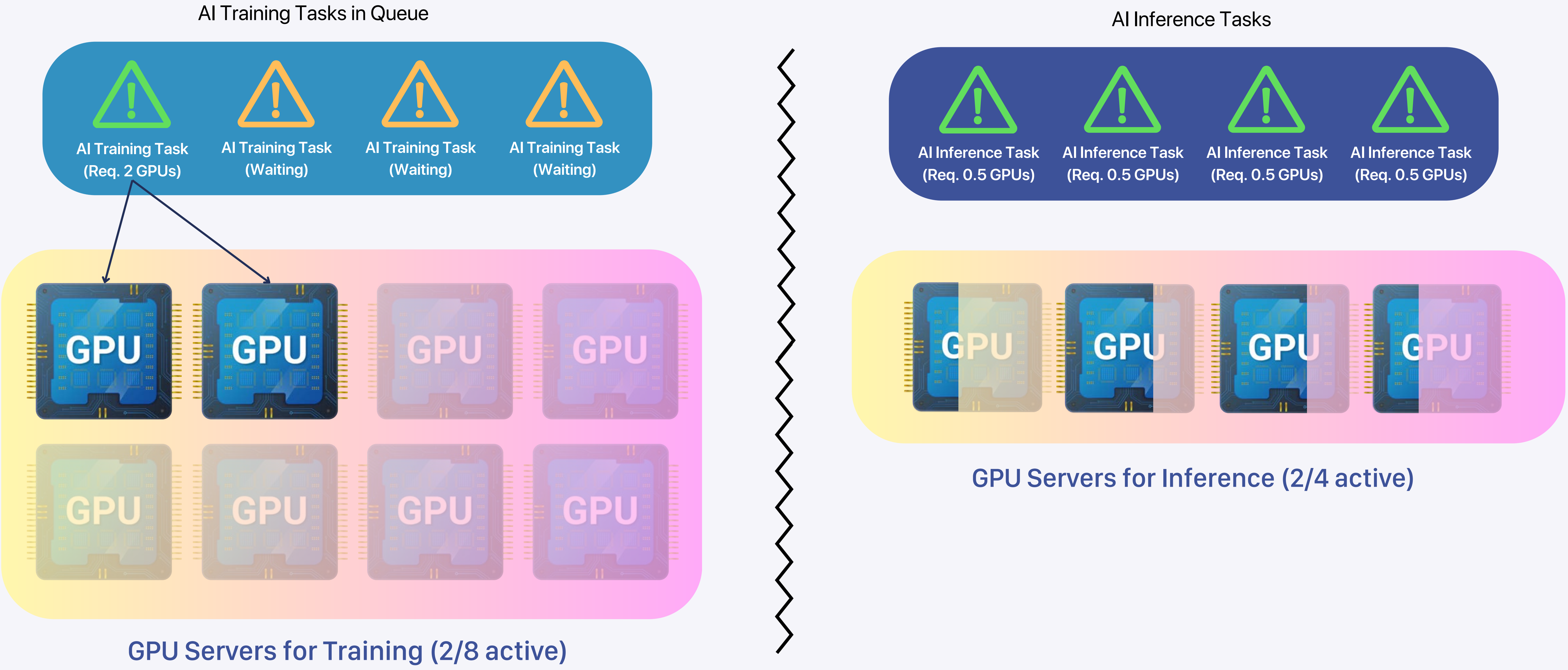
Think of ArcHPC Oracle as an air traffic controller at an airport for instructions; it makes sure that instructions don’t impede each other's execution and this results in an increase in the performance of GPUs reducing compute time and increasing user density from 100% to 300%.

Benefits of ArcHPC Suite

- Maintain processor uptime by memory-level parallelism
- Fine-tuning in the GPU task environment for minimum and maximum compute times at intersection points
- Maximizing the optimal thread arrangement
- Optimizing warp scheduling
- Integrates with job schedulers
- Machine code analysis
- Complementary machine code pairing
- Automated task matching and task deployment
- Understanding of accelerated hardware latencies
- Ability to adjust the performance of tasks in real-time based on business objectives

ArcHPC Interoperability Overview

Common Siloed Infrastructure

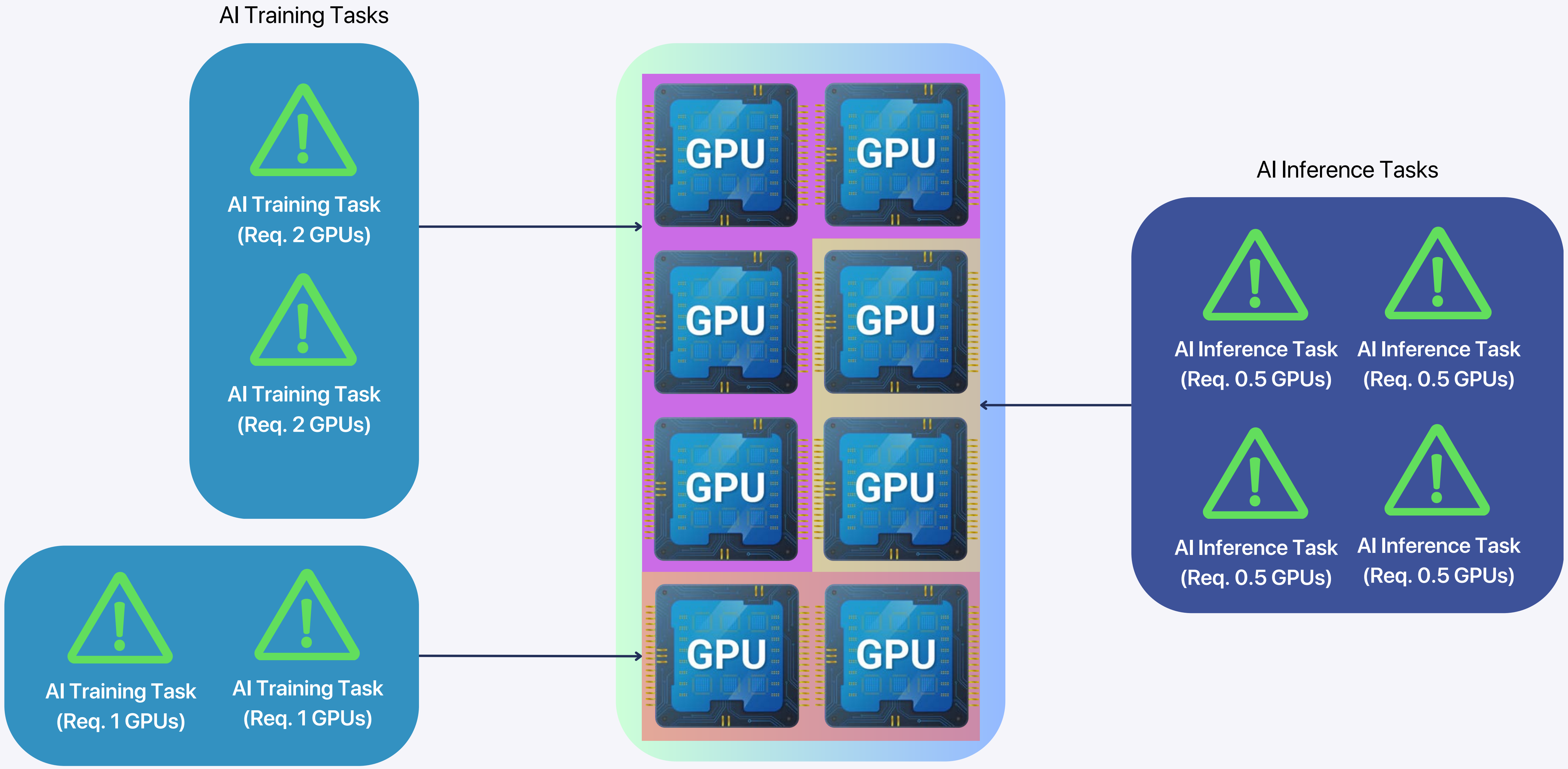


75% GPU Resource Wasted

50% GPU Resource Wasted

Arc HPC Infrastructure

100% GPU Resource Utilized



GPU Servers for Training & Inference (6/6 active)

Applications

Applications

- Operational Performance
  - Thermodynamic Regulation
  - Power Regulation
  - Performance Regulation
  - Priority System
- Accelerated Hardware Cyber Security
- System Control
  - Databinning
  - Pseudo Hardware "Hardware spoofing"
- RTOS / Pre-emptive
- User Defined Governance and Policy Management
- Firmware Kernel

Real World Applications
(Examples)

Real World Applications

RTOS and Performance

- Solar-powered edge devices providing critical information need to maintain operations even during phases or stages where the power supply is limited.
    - In this case for Performance Regulation and Power Regulation, the following abilities are used:
        - [System Discovery, Code Denial and User Defined Rejections].

Accelerating AI Advancement and Performance

- New weapons are detected, and AI defence systems need to be quickly updated to maintain a tactical edge; the AI defence system requires training quickly and moved to top priority.
    - In this case for Performance Regulation, Priority System, the following abilities are used:
        - [Code Trap, Code Replace, Code Intercept, Code Orchestration, System Mapping, System Spoofing and System Compartmentalization].

Cyber Security

- A military or intelligence team wants to inject an AI tool into a system but not be compromised during the connection or duration that it is connected; the security team must secure the system so unauthorized code cannot be run.
    - In this case for Accelerated Hardware Cyber Security, the following abilities are used:
        - [Code Denial, Code Intercept, Code Trap, Code Replace, Code Orchestration, System Compartmentalization and System Spoofing].

Encryption, Cyber Security and Performance

- An intelligence agency wants to inject counterintelligence or bad data by planting a compromised device; they have to ensure the data is believed.
    - In this case for Pre-emptive and Firmware Kernel the following abilities are used:
        - [Code Trap, Code Replace, Code Intercept, Code Orchestration, System Mapping, System Spoofing and System Compartmentalization].

Questions

ARC COMPUTE