GPU Power Variability in ML Systems

Rajesh Shashi Kumar, Ashwin Poduval

1 Introduction

Machine Learning (ML) is a critical workload in contemporary times. ML algorithms are often highly parallelizable and scalable, a property that has been exploited by researchers[1] for over a decade to observe large speedups in performance upon utilizing accelerators such as GPUs. The inherent scalability of ML algorithms has meant that multiple GPUs could be saturated running a distributed ML workload. These trends have led to the development of massive models in the past few years spanning many GPUs and running for an extensive period of time, such as GPT-3[2].

Presently, cloud computing is an extremely popular service used for running such application. Hardware resources can be rented from service providers and the resources used can be adjusted in real time, with cost calculated roughly on the basis of resources used per unit time. However, this model of computation also brings a certain opacity into the picture. The methodology by which resources are allocated is a black box to the customer; they are unaware of whether other workloads could be running simultaneously on a machine issued to them, or if a task was running on the machine immediately prior to the machine assigned to them.

However, such a model of computation also brings scope for interference and heterogeneity in performance. Prior or concurrently running workloads could cause spikes in temperature and power (temporal interference). A workload running on the same rack or machine could also affect the performance by causing variations or spikes in power draw or a change in temperature gradient (spatial interference). Additionally, the scheduling policy could allocate jobs more frequently to some machines, some GPUs more frequently than others, causing wear and tear of these devices. DVFS algorithms could also add to the variability.

In this project, we undertook a study of power variability in multi-GPU settings. We designed a number of cases to study temporal and spatial interference by running combinations of compute-intense SGEMM kernels and noise workloads such as deep neural networks (DNN) and memory-intensive cudaMemcpys. We also ran the SGEMM kernels on a Jetson TX2 board to better understand the effects of DVFS.

Over the course of this project, we hoped to understand the behaviour of multi-GPU systems as would be popular in large-scale GPU deployments. Service providers would desire homogeneity of behavior across these related resources, even as specific expectations may change from job to job or user to user, and even as multiple jobs may use resources on the same host simultaneously. Additionally, they would also wish to recognize if certain units of hardware were being over utilized or worn out to replace such hardware, and also recognize possible scheduling or systemic configuration issues that cause certain devices to become points of failure.

2 Background

There exists an extensive body of prior work studying power and frequency variations in CPUs. There is a growing body of work studying the performance and improving efficiency of GPUs executing ML applications, but the

concept of power variability for such systems is for the most part unexplored. In the rest of this work, we focus on characterization of power variability and defer mitigation to future work because the phenomenon is not easily observable.

2.1 DVFS

Processing elements such as GPUs and CPUs ship with dynamic voltage and frequency scaling (DVFS) algorithms to improve performance efficiency by dynamically adjusting power utilization depending on trends in utilization and operation intensity. However, these algorithms are not perfect, and tend to be reactive. DVFS can also lead to unpredictable behaviour and exacerbate performance heterogeneity; prior works such as [3] discuss the effect of Turbo Boost algorithms which dynamically overclock/boost processor clock speeds in this context.

2.2 Power Variations

Power variations can originate through multiple sources:

- Variations due to manufacturing in properties of device physics such as gate delays and leakage current.
- Node-level variations due to the dynamic interaction between components with high power draw from the same source. These are more prevalent and observable at the system level. We try to characterize the effect of these variation in ML workloads in our work. The dynamic nature of these variations make observability and analysis particularly challenging. DVFS further exacerbates this problem by inducing frequency and voltage changes. Cloud vendors recommend disabling autoboost on GPUs for example to achieve predictable performance at maximum frequency. However, recent generations of GPUs no longer support this mechanism.

2.3 ML Workload Characteristics

ML workloads involve a large number of vector and matrix operations which make them ideal candidates for acceleration using GPUs. ML algorithms also perform better with larger model sizes and data sets. In the internet era, massive amounts of data can be collected fairly easily. However, all this leads to a greater demand for computation, and higher memory pressure, outpacing that available from a single GPU, despite the massive generational performance gains of GPUs and their capacity to launch a huge number of threads. This makes multi-GPU environments necessary for ML workloads. The interaction between different GPUs, and the high utilization of devices in such systems can lead to linked effects leading to power and frequency variations.

3 Experimental Design

In this section, we discuss the design of experiments to observe and characterize power variability in multi-GPU systems under conditions that are hypothesized to reflect the variation. We use the measurements from these experiments to formalize observations into device and application agnostic insights.

Hardware setup We used a two-fold experimental hardware setup:

1. Nvidia Jetson TX2: Controlled system to study effects of DVFS on performance by varying frequency.
2. CloudLab C4130: Target system with 4xV100 GPUs to characterize variation using ML workloads.

Hardware configuration: Jetson TX2 This is a 15-watt single board computer that brings AI computing to the edge [4]. It houses a 256-core NVIDIA Pascal-family GPU with 8GB of system memory and 59.7GB/s of memory bandwidth. An important caveat is that the Jetson does not have dedicated GPU memory. The system memory is shared directly between the CPU and GPU. This implies that the GPU is not limited by PCIe bus speeds and the PCIe management functions of a discrete GPU do not apply.

Hardware configuration: CloudLab We used a C4130 node in the Wisconsin cluster for our experiments. The hardware configuration[5] is captured in Table 1. We found it imperative to understand the hardware layout in the analysis and correlation of variability measurements. A schematic of the rack server is included in Figure 13. Though the hot-swappable redundant power supply unit makes the system resilient to external variations, the dynamic interaction between components at high utilization can possibly still lead to power variability.

CPU	Two Intel Xeon E5-2667 8-core CPUs at 3.20 GHz (TDP:130W each)
RAM	128GB ECC Memory
Disk	Two 960 GB 6G SATA SSD
GPU	Four NVIDIA 16GB Tesla V100 SMX2 GPUs (TDP:300W each)
Power Supply	2x1200W redundant power supply

Table 1: System configuration of a CloudLab C4130 node

Workloads, datasets, models We define two categories of workloads to systematically observe variations:

1. *Base workload*: Hand-written, easily profiled kernel with high compute utilization whose variation will be characterized for analysis.
    - SGEMM kernel using cuBLAS in CUDA. A square matrix of 25536 elements was empirically chosen based on nvprof metrics (Figure 14) reported sm_efficiency of 99.96%
2. *Noise workloads*: External workloads that can induce variations in the base workload through spatial or temporal interference. The intent here was to choose workloads that would replicate allocation scenarios in production systems on large shared clusters. The number of epochs/runtime of noise workloads was carefully chosen to match the duration of execution of the base workload.
    - *RESNET50v1.5*: Single node ML training of vision model, **compute** intensive. We used the Imagenet ILSVRC2012 dataset with a batch size of 128. Source code for model [6].
    - *Transformer*: Multi-GPU data parallel training of language models over NVLink, **memory** intensive. We used the wmt4_en_de dataset with a batch size of 4096 trained over 1-3 GPUs. Source code for model [6].
    - *CudaMemCpy*: Synthetic workload to represent intensive **data movement** over PCIe. Source code [7].

Profiling NVProf [8] was used to monitor and collect system metrics such as temperature, voltage, power and operational frequency. In addition, we captured the kernel runtime for each invocation through manual instrumentation to aid in performance analysis.

Measurements The characterization of variation is dependent on large amounts of data collection. We defined a strategy to account for statistical variations in our data collection to ensure that the variation we observe is indeed power variation and not an artifact of measurement. To do so, we obtained averages for each of the variables with measurements over 100 repetitions and 2-4 sets of these repetitions to account for warmup.

Assumptions We make the following assumptions in our approach:

1. Run profiling on all GPUs to ensure profiling overheads do not interfere with variability measurements.

Mode	Mode Name	Denver 2 Count	Denver 2 Frequency	ARM A57 Count	ARM A57 Frequency	GPU Frequency
0	Max-N	2	2.0 GHz	4	2.0 GHz	1.30 GHz
1	Max-Q	0		4	1.2 GHz	0.85 GHz
2	Max-P Core-All	2	1.4 GHz	4	1.4 GHz	1.12 GHz
3	Max-P ARM	0		4	2.0 GHz	1.12 GHz
4	Max-P Denver	1	2.0 GHz	1	2.0 GHz	1.12 GHz

Table 2: Jetson power modes, frequencies, and core counts

2. V100 does not support controlling AutoBoost. This means that the GPU monitors temperature to dynamically control the operational frequency. We observed overshoot in the power envelope above the rated TDP to nearly 320+25W across runs.

Effects We define a case matrix in section 4, this was centred around effects in two dimensions:

1. Spatial effects: Proximity of GPU relative to each other may introduce variability due to coupling between GPUs in factors like temperature and voltage drop. Systems today are generally not placement aware.
2. Temporal effects: High concurrent utilization may lead to variations due to resource contention in I/O or power

Reproducibility To ensure reproducible results of the experiments presented in this work, we documented the scripts and compiler/tool versions that were used for training/profiling in a git repository [7].

4 Evaluation

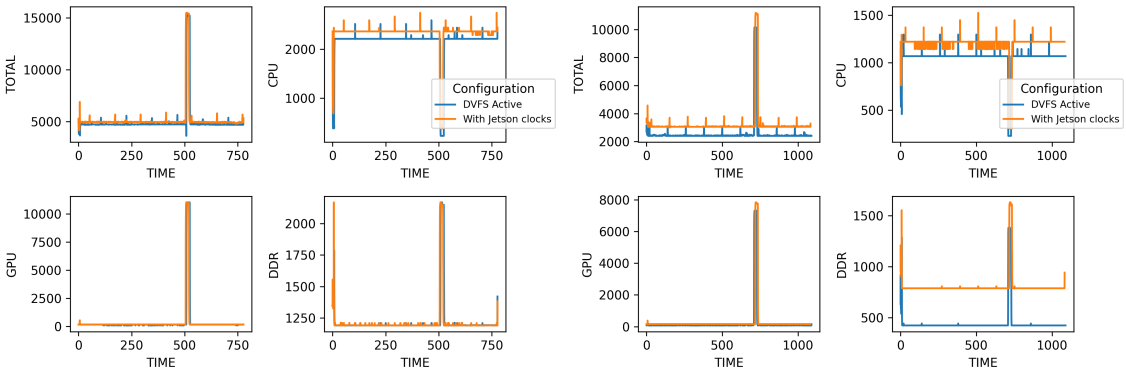
4.1 Evaluating DVFS on a Jetson TX2

We ran the sgemm and dgemm kernels on the Jetson TX2. Square matrices of dimension increasing in powers of 2, from 1 to 16384 were used in the matrix multiplication. These kernels were run twice for each performance mode of the TX2 (which are described in Table 2) - initially for the power model in base configuration, with DVFS active, and then after running the Jetson_clocks.sh script, which disables DVFS by forcing all frequencies to the maximum allowed in the current power state as per the CPU, GPU and EMC (memory controller) governor kernel modules. There are five power models in total, so we ran each experiment 10 times in total (5 power models, and for each power model with and without Jetson clocks.)

For conciseness, we will discuss only the 16k run of sgemm. Since sgemm was also used as the base workload for the CloudLab runs, we felt it pertinent to focus on the corresponding set of observations recorded with the Jetson. Additionally, the majority of the kernels for matrix dimension $<$ 1k were extremely short running, completing in under 2 seconds. The 16k runs were, in comparison, long running, taking 100s of seconds to complete, leading to richer and more conclusive observations from the data.

We have only added the figures for modes 0 and 2 (Fig. 1a and Fig. 1b respectively). The other figures are present in the appendix (Fig.s 15, 16, 17). As can be seen, some of the modes utilize Nvidia's custom Denver2 cores, which are more complex, performant and power hungry than the stock ARM A57 cores. It is evident that mode 0 offers the highest performance, with all cores active and running at max frequencies. As we can see, it completes execution in the least amount of time, taking 750 seconds. However, total power is also higher, with a median power of about 5W and a peak power of 15W irrespective of whether Jetson clocks was active or not.

In contrast, mode 2, which uses all cores, but at lower CPU and GPU frequencies takes 1000 seconds, but uses a sees median total power of about 3-3.5k with Jetson clocks, and around 2-3k without Jetson clocks. It is interesting to note that even when using Jetson clocks, we see spikes in power consumption, and that overall the DVFS algorithm is efficient in most cases, showing a similar amount of variability, consuming slightly lower power, and taking a similar



(a) Mode 0 (b) Mode 2

Figure 1: SGEMM kernel power consumption for power modes 0 and 2. Power in mW, time in sec.

amount of time. We noticed an exception in case of mode 1, where Jetson clocks seemed to consume less power on average, and we don't quite understand how this could have occurred - we hypothesize this could have resulted from a background process starting up, we did not take multiple runs to filter out variation due to randomness.

4.2 Characterization of variability in a multi-GPU system

This section describes the analysis from experiments run on the CloudLab multi-GPU node described in section 4. The characterization of variability is non-trivial given the large number of variables, hence causation is a challenging task at the system-level. To address this, we first establish a series of intents to test our hypotheses on the plausible effects of variability. We then attempt to correlate the results against the intents to see if we can derive formalized insights. The experimental cases for the intents are described in Figure 2 and are intended to model real world usecases to observe the following:

1. Influence of (n-1) GPUs on the Base workload
2. Model time and space locality in a cloud environment
3. Spatial effects with same workload but different GPU allocations.

The intents shaded in blue indicate repetition for a different noise workload. As an example of the orchestration of these experiments, a snippet of the nvidia-smi output for Case 19 where SGEMM and two noise workloads (RESNET+Distributed Transformer) are simultaneously active is shown in Figure 18. In the remainder of this section, we use the case numbers to reference experiment scenarios indicated in Figure 2.

Case	Intent	GPU1	GPU2	GPU3	GPU4
1	Effect of increasing number of GPUs on base workload	SGEMM			
2		SGEMM	SGEMM		
3		SGEMM	SGEMM	SGEMM	
4		SGEMM	SGEMM	SGEMM	SGEMM
5	Effect of increasing number of GPUs on noise workloads	RESNET			
6		RESNET	RESNET		
7		RESNET	RESNET	RESNET	
8		RESNET	RESNET	RESNET	RESNET
9 (4)	Effect of homogenous workload on each other (Time locality)	SGEMM	SGEMM	SGEMM	SGEMM
10	Spatial effects (1): Single source of noise	SGEMM	SGEMM	SGEMM	RESNET
11		RESNET	SGEMM	SGEMM	SGEMM
12	Spatial effects (2): Alternating separation in space of noise sources	SGEMM	RESNET	SGEMM	RESNET
13		SGEMM	SGEMM	RESNET	RESNET
14	Spatial effects (3): Does GPU allocation in a cluster impact performance?	SGEMM	RESNET	RESNET	RESNET
15		RESNET	SGEMM	RESNET	RESNET
16		RESNET	RESNET	SGEMM	RESNET
17		RESNET	RESNET	RESNET	SGEMM
18	Effect of distributed training on base	SGEMM	Transformer		
20	Effect of data movement on base	SGEMM	MEMCPY	MEMCPY	MEMCPY
21		SGEMM	SGEMM	MEMCPY	MEMCPY
22		SGEMM	SGEMM	SGEMM	MEMCPY
23	Manually choose best GPU in terms of idle temp to see if it performs better?		SGEMM		
24	Spatial effects (1): Single source of noise	SGEMM	SGEMM	SGEMM	Transformer
25		Transforme	SGEMM	SGEMM	SGEMM
26	Spatial effects (2): Alternating separation in space of noise sources	SGEMM	ransforme	SGEMM	Transformer
27		SGEMM	SGEMM	Transformer	
28	Spatial effects (3): Does GPU allocation in a cluster impact performance?	Transforme	SGEMM	Transformer	
29		Transformer		SGEMM	Transformer
30		Transformer			SGEMM
19	Effect of heterogeneous utilization on base workload	SGEMM	RESNET	Transformer	
31		Transformer		SGEMM	RESNET
32		SGEMM	Transformer		RESNET
33	Proving that temperature effect due to spatial proximity affects performance (runtimetime/frequency)			SGEMM	
34				SGEMM	SGEMM
35			SGEMM	SGEMM	SGEMM
Case	Intent	GPU1	GPU2	GPU3	GPU4

Figure 2: Intent for variability experiments

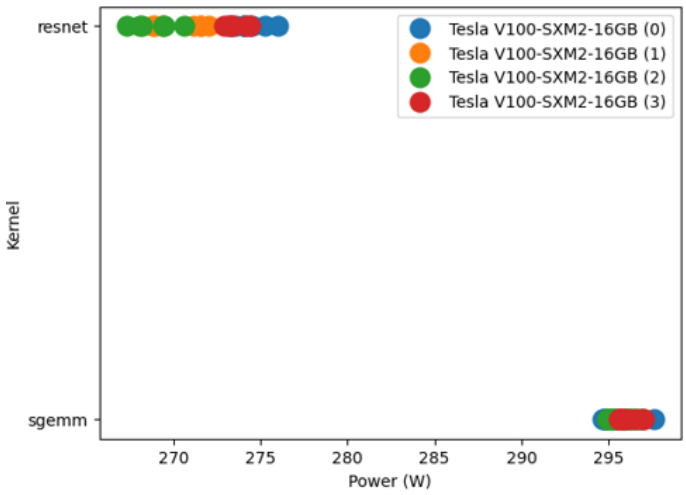


Figure 3: Average power for SGEMM+RESNET cases 10-17

4.2.1 Average power consumption (RESNET50 + SGEMM)

Figure 3 illustrates the average power draw for the base SGEMM workload measured across cases 10-17 where RESNET is the noise workload. Both RESNET/SGEMM were observed to be at 100% utilization, however the average power draw is lower for RESNET due to the processing between iterations in an epoch. There is also more variability in power draw for RESNET likely due to the same reason. Patterns such as GPU0 being the consistent outlier indicate that variation is a observable and measurable phenomenon.

Takeaway #1: Variability in power exists both within a workload and between workloads.

4.2.2 Average kernel runtime for base (SGEMM) workload

Figure 4 illustrates the average kernel runtime for the base SGEMM workload measured across all cases. There is a **3.5% variation** observed in kernel runtime across cases. This could possibly be exacerbated for long running workloads. In addition, a distinct pattern emerges in this plot indicating the performance heterogeneity for the same workload across GPUs. In addition, we observe (i) GPU0 being the worst performer and (ii) approximate hints of co-location in placement of GPUs on the node. GPU0 being the worst performer can be attributed due to the fact that most single-GPU workloads are scheduled on this device ID by default making it prone to thermal wear. This is evident from higher idle temperatures for GPU0 as seen in Figure 5.

Takeaway #2: Variability is the likely cause of observable performance heterogeneity of the base workload across GPUs on the same node.

4.2.3 Effect of noise on base workload

Figure 6 illustrates the contrast between case 1 where SGEMM alone is active and case 14 where SGEMM is co-located with RESNET on other GPUs. Comparing 6 (a) and 6 (c) reveals two effect of the noise workload on the base

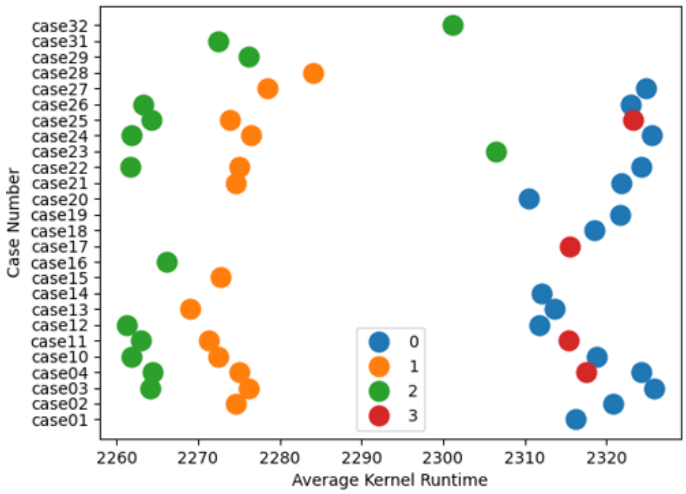


Figure 4: Average kernel runtime for BASE(SGEMM) workload

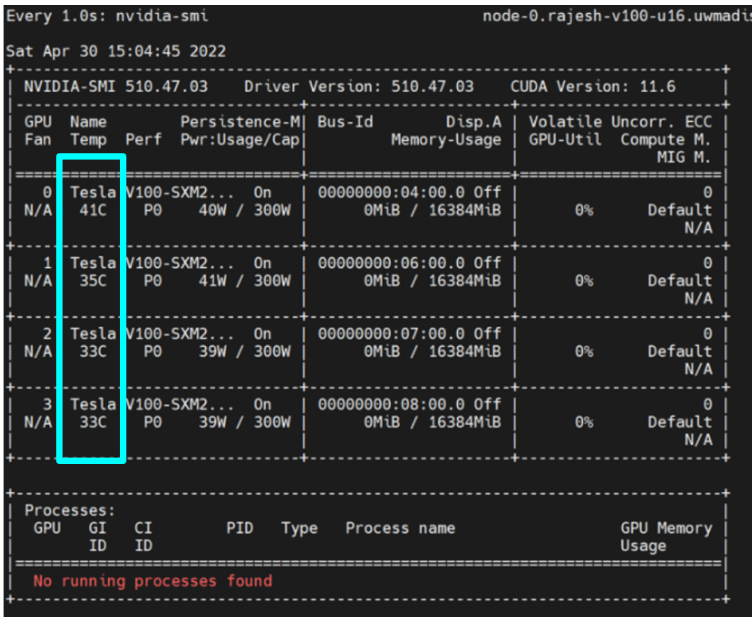


Figure 5: Idle temperature across GPUs

workload. First, there is increase power draw variation due to RESNET's volatile power consumption changes. Second, the SGEMM workload reaches higher temperatures in the presence of the noise workload. Also, the coupling in power variation between GPU2,GPU3 and GPU1,GPU0 indicate that they share proximity in placement on the rack server. We further confirm the intuition on placement of GPUs on the rack server in section 4.2.5.

Takeaway #3: Spatial effects from proximity in placement can induce variability in power and higher operating temperature in the base workload.

Furthermore, we observed that RESNET's volatile power draw pattern on GPU2 (Figure 7a) results in bigger steps

Figure 6: Time-series power and temperature plots for Case 1 and 14

in frequency when contrasted with SGEMM running on GPU1 (Figure 7b). Given that RESNET's power consumption is about 270W on an average from 3, the frequency step can be attributed to the room in power envelope for higher boosts through DVFS.

Takeaway #4: Frequency steps show a direct correlation with the magnitude of power variation.

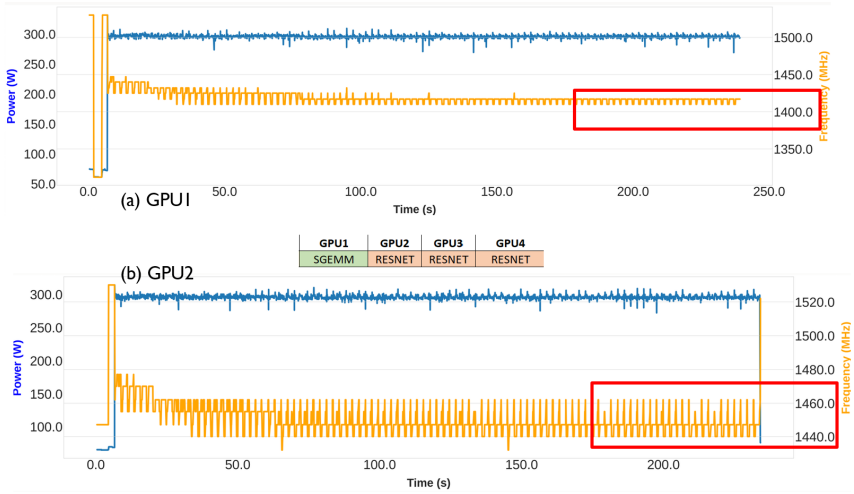


Figure 7: Time-series power and frequency plots for Case 14

4.2.4 Effect of data movement on training: What is volatile utilization?

Nvidia-smi is frequently used to monitor GPU attributes such as temperature, power and volatile utilization. While intuitively it might seem that 100% volatile utilization would translate to reaching the power limit, we found this is

not the case. In case 20 of our experiment to study the effect of data movement on the base workload, we noticed that the cudamemcpy workload reached a 100% utilization with minimal power draw as illustrated in Figure 8. On further investigation, we found out that volatile utilization is simply a measure of a kernel executing on the GPU but does not indicate the number of active SMs (as captured by achieved_occupancy) which determines dynamic power consumption. This introduces an interesting case for space sharing where jobs of differing attributes can be scheduled on the same GPU to achieve better resource utilization.

Takeaway #5: Nvidia-smi volatile utilization does not have correlation with power consumption. The nvprof achieved_occupancy metric is a better indicator of estimated power consumption.

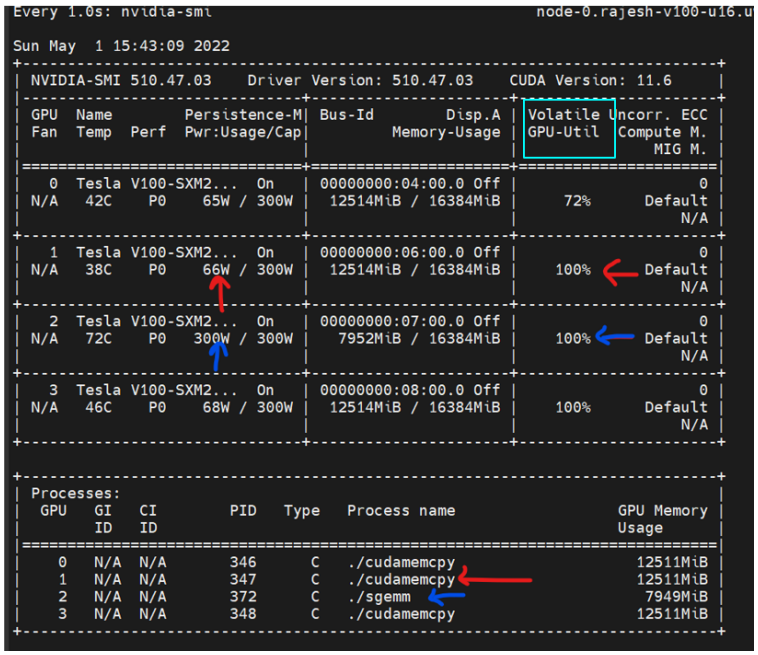


Figure 8: Measure of volatile utilization for Case 20 from Nvidia-smi snippet

4.2.5 Spatial effects: Does GPU allocation in a cluster impact performance?

Cloud GPU providers allocate GPU to users in an opaque manner. Multiple users can be sharing different GPUs on the same node. Performance predictability and efficiency are desirable in long running ML workloads that are expensive in terms of compute power. In this section, we attempt to prove with experimental results that specific GPU allocation can impact application performance.

While running a single-GPU experiment, we interestingly observed that an active-GPU3 was inducing a significant increase in temperature of idle-GPU4 as shown in Figure 9. This did confirm the existence of spatial effect on idle neighbouring GPUs but the performance implication was not directly evident. We formulated 3 new cases (33, 34, 35) in Figure 2 to investigate this further. Our hypothesis was two fold:

1. GPU3 possibly induces a performance slowdown in GPU4 because of the spatial proximity from the earlier observation of induced increase in temperature (Case 34, Case 35)
2. At the same time, GPU2 should not have measurable performance impact since it is not an immediate neighbor (Case 35)

Figure 10 shows that GPU3 adversely affected the performance (observable both as frequency in 10a and runtime impact in 10b) of GPU4 while it had no effect on GPU2. Similarly, Figure 11b reinforces the observation in 9 that

GPU3 induces a temperature increase on GPU4, thereby throttling power draw on GPU4 as seen in Figure 11a.

Takeaway #6 Neighbor active-GPUs can impact application performance as a consequence of induced variability. This results in performance heterogeneity across GPUs on the same node, thus placement and activity aware GPU allocation can perceivably improve application performance.

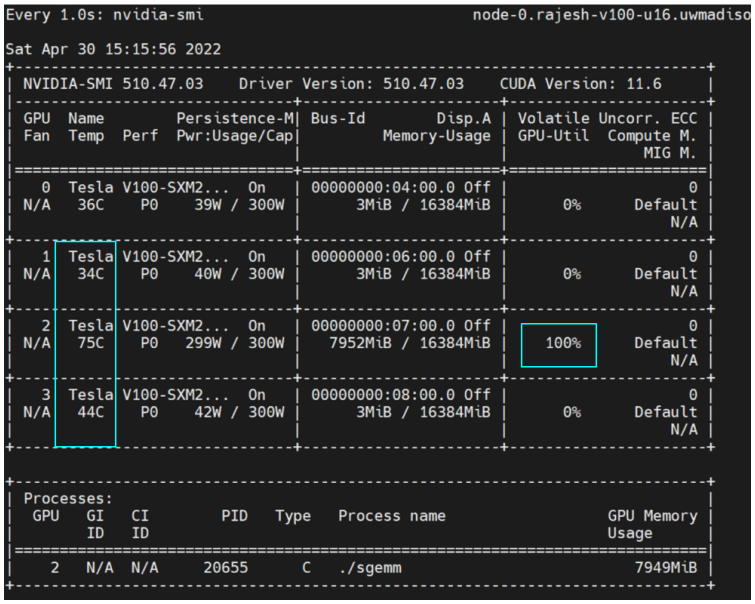
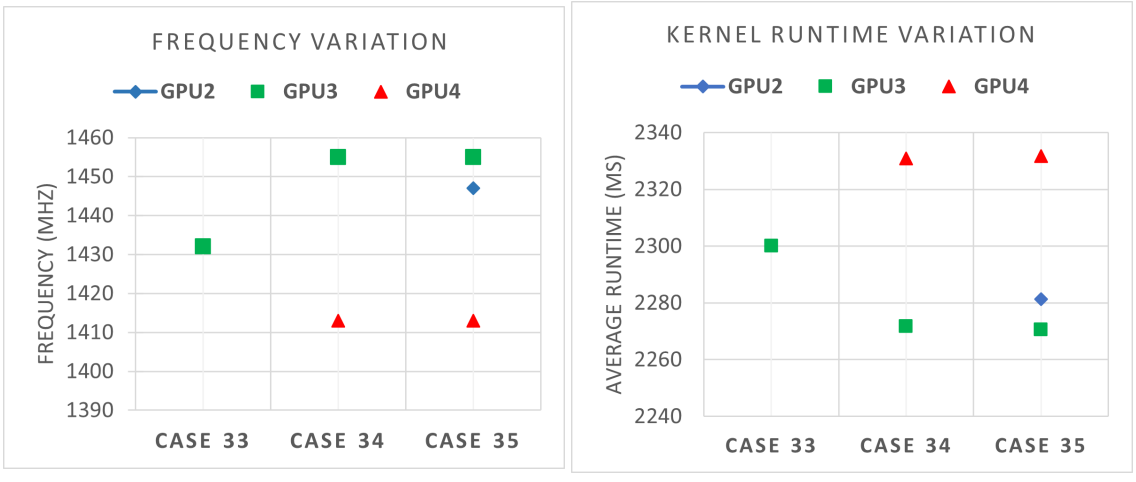


Figure 9: Case 20 indicating higher temperature on GPU4 induced by GPU3 activity



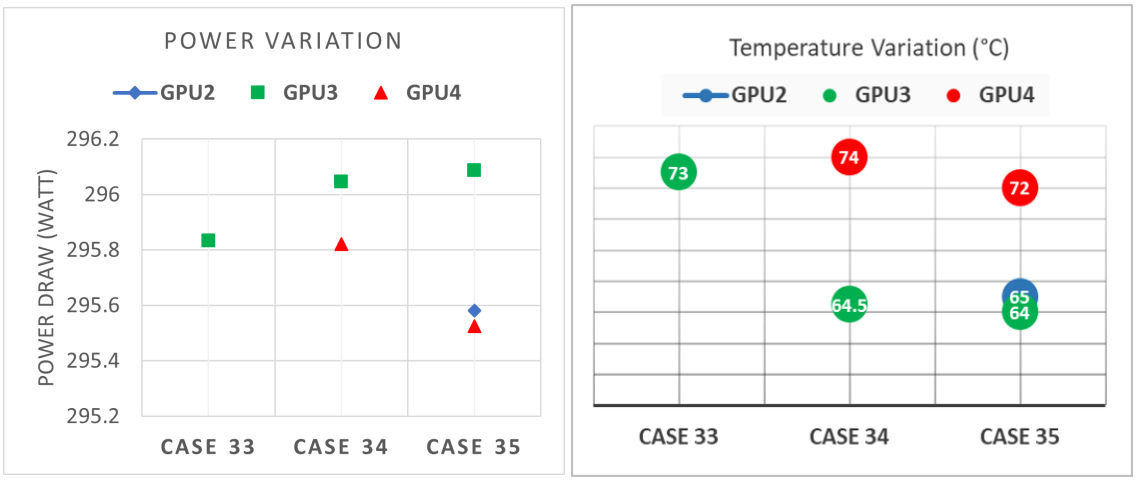
(a) Frequency plot (Higher is better)

(b) Kernel Runtime (Lower is better)

Figure 10: Case 33, Case 34 and Case 34 to prove performance impact due to GPU allocation

4.2.6 Effect of increasing number of GPUs on base workload

We use cases 1-4 to study the effect of concurrent execution of the base workload on all GPUs. In our observation, this introduced a correlation in Figure 12 similar to that seen in Figure 4. Using the correlation across cases, we arrived at the following prediction for the placement of GPUs on the server for the schematic presented in Figure13:



(a) Power plot (Higher is better)

(b) Temperature Plot (Lower is better)

Figure 11: Case 33, Case 34 and Case 34 to prove performance impact due to GPU allocation

—GPU2,GPU3,GPU0,GPU1—. Unfortunately, we could not truly verify this with the actual configuration on the server since we did not have physical access to the node.

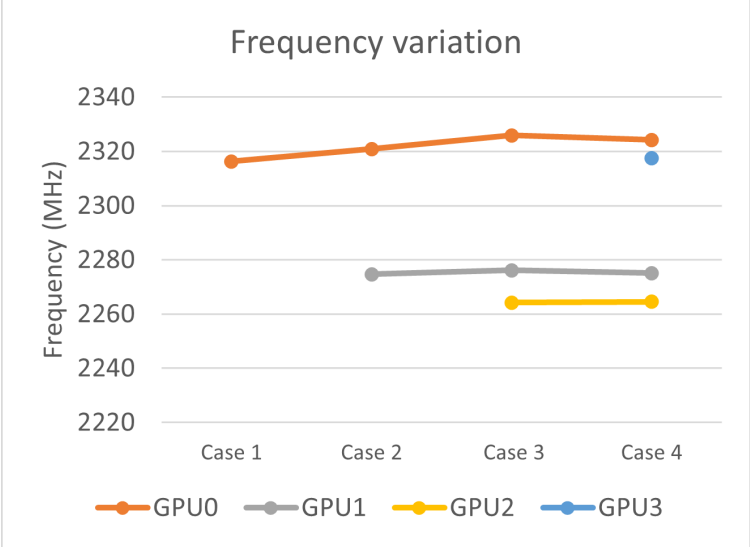


Figure 12: Kernel runtime across cases 1-4

5 Related Work

CPU Performance variability in HPC systems is extensively discussed by Acun[9]; this work was our main point of reference. This dissertation provides analysis on frequency, temperature and power variations in HPC systems using thousands of cores and applications and discusses mitigation techniques. The author also discusses runtime mitigation of application level variation, i.e. variation originating from the characteristics of the application.

Related works by Acun et al. are [10], [11], [3], [12], [13]. Frequency, Power and Temperature variations due to Turbo boost dynamic overclocking algorithms in processors is discussed in [10]. An adaptive runtime system

for controlling power consumption by intelligent resource management and scheduling, and reliably controlling temperature is presented in [11]. [3] demonstrates the efficacy of performing fine-grained DVFS on a per-core basis with respect to power consumption.

System level optimizations to take advantage of the properties of recommendation models are illustrated in [12]. Finally, [13] studies and mitigates efficiency problems in datacenter GPU deployments by profiling, collecting traces and performing analysis on the collected data.

6 Future work

Possible directions for future work:

- We find the set of experiments presented in Figure 2 to be an useful stressmark on a single node to characterize variability. One direction for future work would be analysis of a targeted subset of these experiments across a number of nodes in a cluster to validate the insights in our work.
- Investigation of continuous profiling mechanisms in production systems as opposed to running stressmark to characterize GPU performance that impacts availability. We found that Nvidia has started work in this direction of background telemetry through the Data Center GPU manager project to monitor system factors such as power, frequency to facilitate GPU health monitoring and efficient cluster scheduling policies. However, they do not take into account the effects of inter-GPU variability and spatial effects which we found lacking.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[3] B. Acun, K. Chandrasekar, and L. V. Kale, "Fine-grained energy efficiency using per-core dvfs with an adaptive runtime system," in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8, 2019.

[4] NVIDIA, "Jetson TX2." https://developer.nvidia.com/embedded/jetson-tx2, 2022.

[5] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, pp. 1–14, July 2019.

[6] Nvidia, "Deep Learning Examples." https://github.com/NVIDIA/DeepLearningExamples. Accessed: 2022-05-12.

[7] R. Shashi Kumar and A. Poduval, "Power Variability in ML Systems." https://github.com/rajesh-s/mlsys-gpu-power-variability. Accessed: 2022-05-12.

[8] NVIDIA, P. Vingelmann, and F. H. Fitzek, "CUDA, release: 11.6." https://developer.nvidia.com/cuda-toolkit, 2022.

[9] B. Acun, "Mitigating variability in HPC systems and applications for performance and power efficiency." https://www.ideals.illinois.edu/bitstream/handle/2142/99502/ACUN-DISSERTATION-2017.pdf?sequence=1&isAllowed=y.

[10] B. Acun, P. Miller, and L. V. Kale, "Variation among processors under turbo boost in hpc systems," in *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[11] B. Acun, A. Langer, E. Meneses, H. Menon, O. Sarood, E. Totoni, and L. V. Kalé, "Power, reliability, and performance: One system to rule them all," *Computer*, vol. 49, no. 10, pp. 30–37, 2016.

[12] B. Acun, M. Murphy, X. Wang, J. Nie, C.-J. Wu, and K. Hazelwood, "Understanding training efficiency of deep learning recommendation models at scale," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 802–814, 2021.

[13] L. Wesolowski, B. Acun, V. Andrei, A. Aziz, G. Dankel, C. Gregg, X. Meng, C. Meurillon, D. Sheahan, L. Tian, J. Yang, P. Yu, and K. Hazelwood, "Datacenter-scale analysis and optimization of gpu machine learning workloads," *IEEE Micro*, vol. 41, no. 5, pp. 101–112, 2021.

Appendices

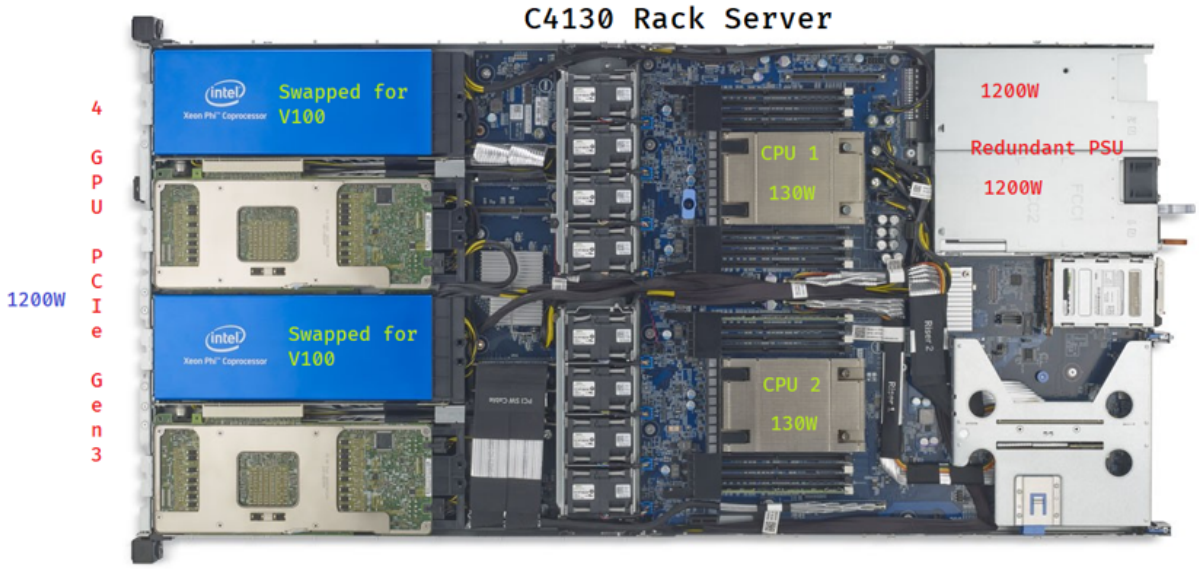


Figure 13: C4130 rack server schematic

```
rajeshwi@node-0:/store/sgemm-cuda$ sudo -E env PATH=$PATH  nvprof --print-gpu-trace  --kernel-latency-timestamps on  --device-buffer-size 128 --continuous-sam
pling-interval 1 --metrics sm_efficiency,achieved_occupancy,sysmem_utilization -f ./sgemm 25536 1 0
==23387== NVPROF is profiling process 23387, command: ./sgemm 25536 1 0
Filename: host_A_25536.binSize:2608349184
Filename: host_B_25536.binSize:2608349184
==23387== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Replaying kernel "volta_sgemm_128x128_nn" (done)
Kernel 0 Runtime = 4772.12
==23387== Profiling application: ./sgemm 25536 1 0
==23387== Profiling result:
        Device   Context    Stream              Kernel  sm_efficiency  achieved_occupancy  sysmem_utilization
                                                                    %
Tesla V100-SXM2         1        16  volta_sgemm_128x128          99.96            0.249937            Low (1)
```

Figure 14: Nvprof metrics for SGEMM base workload

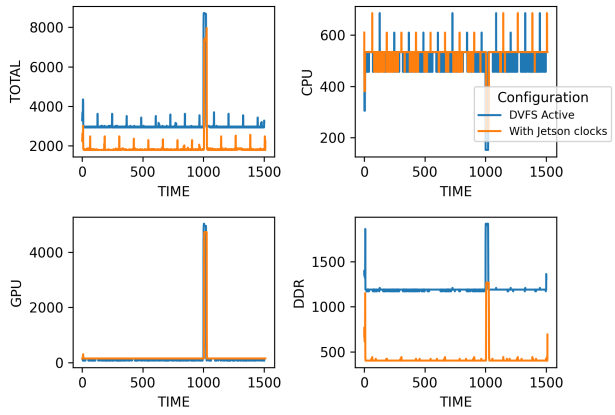


Figure 15: Mode 1

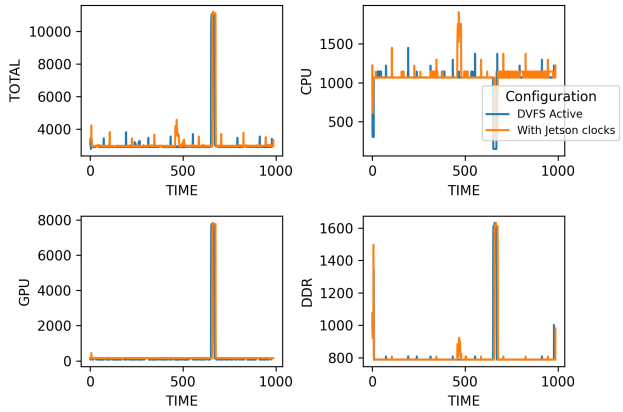


Figure 16: Mode 3

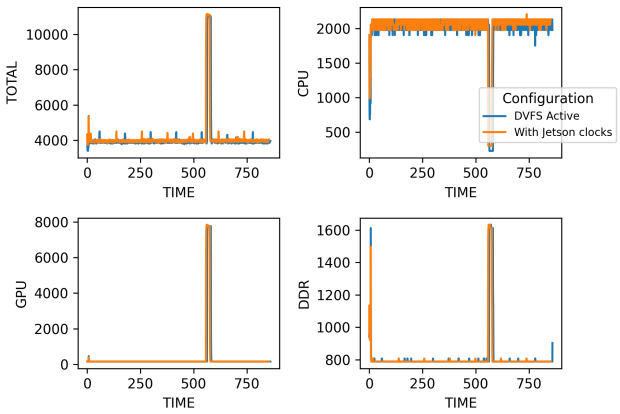


Figure 17: Mode 4

```
Every 1.0s: nvidia-smi

Sun May  1 13:04:55 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:04:00.0 Off |                    0 |
| N/A   58C    P0   297W / 300W |   7900MiB / 16384MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   49C    P0   276W / 300W |  13424MiB / 16384MiB |     99%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   60C    P0   172W / 300W |  10588MiB / 16384MiB |     95%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:08:00.0 Off |                    0 |
| N/A   65C    P0   178W / 300W |  11210MiB / 16384MiB |     94%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      8372      C   ./sgemm                          7897MiB |
|    1   N/A  N/A      8369      C   python                          13419MiB |
|    2   N/A  N/A      8143      C   /opt/conda/bin/python           10585MiB |
|    3   N/A  N/A      8144      C   /opt/conda/bin/python           11207MiB |
+-----------------------------------------------------------------------------+
```

Figure 18: nvidia-smi output for case 19