CoFRIS: Coordinated Frequency and Resource Scaling for GPU Inference Servers

Marcus Chow
Department of Computer Science and Engineering
University of California, Riverside
Riverside, USA
mchow009@ucr.edu

Daniel Wong
Department of Electrical & Computer Engineering
University of California, Riverside
Riverside, USA
danwong@ucr.edu

***Abstract*—Data centers have a variety of metrics that they must adhere to. Not only do they have to meet the rate of incoming requests, but each request also has a service level objective (SLO) that they must satisfy. However, the *average latency* of a single request typically is much faster than the *tail latency* of the SLO. This creates a *latency slack* gap between the average and tail latencies. This latency slack can be exploited to reduce power by slowing down requests through a variety of techniques, such as frequency and resource scaling. However, we show that in an inference server context, frequency alone cannot slow down a request far enough, leaving slack left to be explored. To make up this slack, we propose CoFRIS, a coordinated frequency and resource scaling effort for GPU inference servers. CoFRISdynamically configures the GPU frequency and active resources to minimize power while meeting variable throughput and latency demands. We evaluate CoFRISwith compute unit (CU) level power gating and improve power consumption by 28% over no frequency or resource scaling, 13% improvement over using only frequency scaling, and 5% over using only CU resource scaling.**

***Index Terms*—GPU Inference Server, Resource Scaling, Frequency Scaling, Power Gating**

I. Introduction

Data centers have increasingly adopted the use of GPUs to accelerate Machine Learning (ML) and Inference-as-a-Service workloads [12], [14], [30], [33]. However, prior works have shown that a single inference request typically under-utilizes GPU resources [17], [21]. While GPU resource utilization can be improved through larger batch sizes, care must be taken to ensure Service Level Objectives (SLO) are not violated, which forces a GPU to process smaller batches [18]. As SLOs are targeted for the tail or 99%-tile latency, there is a large *latency slack* between the average response time and tail latency, as illustrated in Figure 1. This *latency slack* can be exploited to save power by employing different techniques to push the average latency closer to the tail (red distribution in Figure 1), such as using DVFS [6], [25], [28]. While DVFS has been shown to be an effective method for reducing power, it might not be able to completely bridge the latency slack. This limitation is due to the limited number of frequency states in the hardware, as well as the range of the states.

Another technique that has been used is *resource scaling* within a single GPU [26], [27], [29]. *Resource scaling* takes advantage of the fact that a workload may under-utilize the GPU and therefore does not need all of the available resources. We can use this to our advantage to save power, by coordinating resource and frequency scaling we can close the slack gap and reduce power even more (blue distribution).

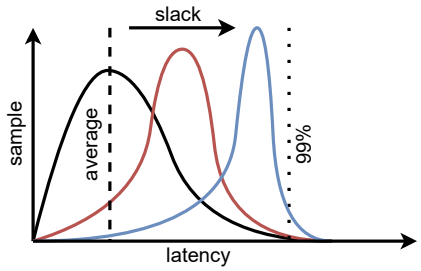


Fig. 1: Black distribution describes the slack between average and tail (99%) latency. Red distribution illustrates how isolated scaling can leave gap to be exploited. Blue distribution outlines our work to close the left over gap through cooridanted frequency and resource scaling.

Towards this end, we present CoFRIS, a framework to coordinate frequency scaling and resource scaling to improve the energy efficiency of GPU inference servers. Our paper makes the following contributions:

- Investigate frequency and resource scaling characteristics of modern GPU hardware.
- Characterize a variety of inference workloads sensitivity to frequency and resource scaling, with respect to latency, throughput, and power consumption.
- Propose CoFRIS , a runtime that minimizes GPU power consumption coordinating frequency and resource Scaling without violating SLO.
- We demonstrate that CoFRIS can achieve 28% average power reduction compared to the baseline.

II. Background

GPUs are massively parallel architectures that can process thousands of threads concurrently. GPUs consist of multiple Compute Units (CUs) where each can process up to 2,560 threads in groups of 64 (AMD) or 32 (Nvidia) threads, called a wavefront. These compute units are organized into *clusters*, called Shader Engines (SEs). In our experiments, we use an

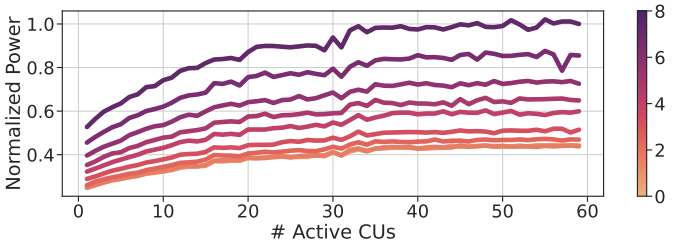


Fig. 2: Power trends of resource (x-axis) and frequency scaling (color bar) ***as measured on a real AMD MI50 GPU***. While resource scaling can reduce power consumption, current AMD GPUs do not automatically cause CUs to be gated due to limitations from AMD.

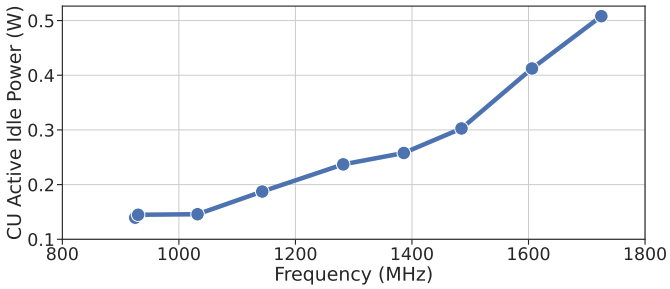


Fig. 3: Calculated per-CU active idle power for each frequency. Higher frequencies lead to higher power savings through gating.

AMD MI50 GPU, that organizes 4 SEs of 15 CUs each, with a total of 60 CUs. Through AMD's CU Masking API [2] and the ROCm System Management Interface (ROCm SMI) Library [1], CU resources and frequency can be *scaled*, as well as measuring the GPU's total power. AMD's MI50 has 9 frequency steps ranging from 925MHz to 1725MHz. We refer to each combination of frequency step and number of active CUs as the GPU's *configuration space*. Although the CU Masking API gives us control over which CUs are active, previous work have shown that how active CUs are distributed across SEs have a significant impact on performance and power consumption [8]. Due to this, for our experiments, we employ a *conserved* policy, which first finds the minimum number of SEs that satisfy the number of active CUs. Then, it evenly distributes the CUs across that subset of SEs. This policy has been shown to avoid any load imbalance from round-robin thread block scheduling.

A. Characterising Frequency/Resource Scaling in AMD GPUs

To highlight the power properties of frequency scaling and resource scaling on inference servers, we sweep a range of frequency levels and resource scaling levels while running our suite of inference workloads (more details on workloads in our evaluation section). Figure 2 shows the geometric mean of these results, with the x-axis indicating the number of active CUs, the color bars indicating the 9 frequency scaling steps, and the y-axis indicating the GPU's power normalized to the maximum observed power consumption.

Across all frequency steps, we observe that the hardware does not activate any CU or SE power gating as shown by the relatively constant power when scaling CUs from 60 to 32. Then, at 31, there is a dip in power which is most pronounced at the highest frequencies, indicating potential power gating is activity, allowing for most of the power savings to be seen when scaling from 30 to 1 active CU.

However, the amount of power savings achievable through resource scaling appears limited compared to the amount of power savings achievable through frequency scaling. This implies that *resource scaling* may be ineffective, compared to frequency scaling, in saving power in current hardware.

Our experimental results directly contradict prior works that have demonstrated effective CU-level power gating in real AMD GPUs through internal, firmware-level tools. [26], [27]. **We discovered that this is because AMD's CU Masking API *does not* automatically cause the CUs to be power gated** [3]. In fact, these CU power gating features have no publicly available control to end users, even though the hardware does have internal CU-level power gating features. Therefore, we as researchers, are not able to reproduce these effects as demonstrated by AMD. ***In our work, to evaluate the potential benefits of resource scaling, we will model both CU and SE power gating granularities to evaluate power gating due to resource scaling as though we have internal AMD tooling.***

B. Modeling Power Gating Savings in AMD GPUs

To estimate power savings that are available through internal AMD tools but not consumers, we need to estimate the amount of power-gating savings per CU. To model possible power saving by power gating individual CUs, we first need to find how much power a single CU consumes while in *active idle* or $CU_{ActiveIdle}$, by using the following equation.

$$CU_{ActiveIdle} = (GPU_{ActiveIdle} - GPU_{Idle})/(N_{CU})$$

This includes the power while the CU is being clocked but not actively running anything and the static leakage power. We measure GPU_{Idle} power, which is the power of the GPU while nothing is running to be 15W on the MI50. Next, to find $GPU_{ActiveIdle}$, we continuously launch empty kernels of 1 warp wide for a set amount of time and record $GPU_{ActiveIdle}$. Because we are launching empty kernels, this ensures that we are not potentially measuring any extra dynamic power from a CU or the GPUs memory system. Dividing the total active idle power by the number of CUs gives us the per CU active idle power.

We discover that this active idle power differs for each frequency as shown in Figure 3. We speculate that this may be due to power gating not just saving static energy, but some form of dynamic energy, such as dynamic clock gating where more power is consumed with higher frequency. We see that the greatest amount of power savings can come from running at higher frequencies as leaving a CU ungated would result in high unused power, which is consistent with previous intuition

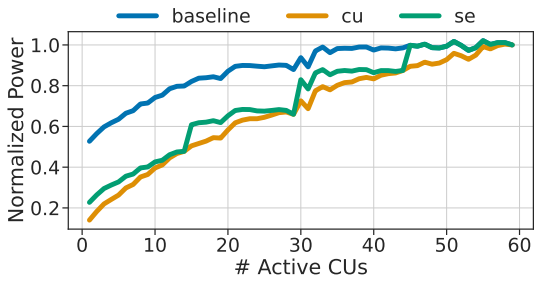


Fig. 4: Calculated power savings from power gating at CU and SE granularity. Per CU Gating saves on average 9% over SE gating.

of multi-core scaling and the need for power gating unused areas of the chip [11]. Using the calculated per CU idle power, we can then model CU level power gating in the whole chip by multiplying $CU_{ActiveIdle}$ with the number of inactive CUs and subtracting that from the measured GPU power.

CU vs SE granularity power gating. While area overheads are a significant consideration in chip design, in our work, we aim to characterize how coordinating frequency and resource scaling can reduce power usage regardless of gating granularity. This is why we evaluate potential power savings at CU and SE gating granularities in Figure 4. Here, the baseline is the power trend of resource scaling at max frequency (the top power curve in Figure 2). By subtracting our modeled per CU active idle power from the baseline, we can see how the granularity of power gating affects power savings. At the SE granularity, we see distinct steps in power savings corresponding to a SE being gated only when all of its CUs are inactive. Whereas, if we gate at the CU granularity, we get a near-linear reduction in power. In total, only when power gating is invoked, we can see the benefits of resource scaling in the GPUs. On average, per CU gating can save an additional 9% over SE level gating.

III. CoFRIS: Frequency and Resource Coordination at Runtime

In this section, we introduce CoFRIS, a runtime to coordinate frequency scaling and resource scaling to close the latency gap in GPU-based inference servers. CoFRIS aims to minimize the power consumption of inference servers by finding the ideal trade-off between frequency and resource scaling while satisfying the QoS requirement for varying incoming RPS rates. To highlight the functionality of CoFRIS, we will first present a characterization study of the impact of frequency and resource scaling on inference workloads. Then we will present a framework that coordinates frequency and resource scaling to enable energy-efficient GPU inference serving.

A. Characterizing Frequency and Resource Scaling for Inference Workloads

We now characterize how coordinated frequency and resource scaling affects inference workloads. In Figure 5 we sweep through the GPUs frequency and resource configuration while measuring the latency, throughput (RPS), and power for various inference models.

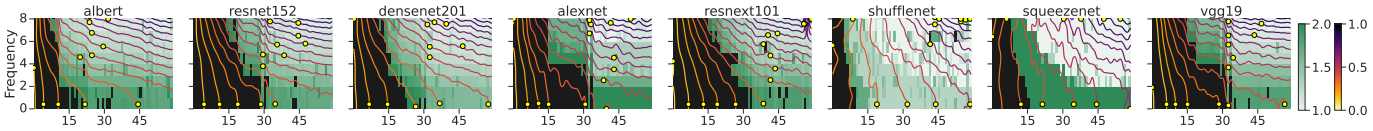
Impact on latency. The impact on tail latency is shown in the background of each figure as a green heatmap, ranging from 1x to 2x. SLO is chosen as 2x the latency of the model when running at max frequency and full resources, which is similar to the methodology used in previous works [5], [8], [20]. Later, we will perform a sensitivity analysis with varying SLO levels. For now, any configuration that does not meet the SLO is shown as a black box in the heatmap and is taken as an invalid configuration but is shown for completion.

We see that scaling down frequency increases latency at a faster rate than resource scaling. (It gets darker green quicker going top to bottom than right to left.) However, even at the slowest frequency we only start to see SLO violations when we additionally scale down to 30 or 15 CUs, depending on the tolerance of the model. This shows that frequency scaling alone is not able to fully exploit the large latency slack between average and tail latencies. Only with coordination between both frequency and resource can we squeeze as much slack as possible for energy savings.

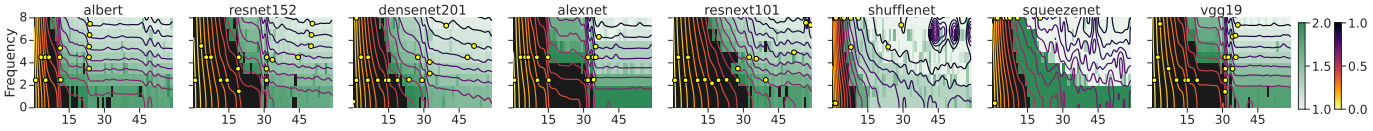
Impact on power and RPS. For Figure 5a, we show the power consumption with CU granularity power gating of each configuration and interpolated it as contour lines to better visualize the trend. The colors of the contours are normalized to the top right configuration. In Figure 5b, we measured the max throughput achieved for each configuration with the highest RPS, which, again, would be the top right of each figure. Each contour can be seen as a Pareto front, as moving along the curve will not provide any power or RPS benefits, respectively. However, we need to consider that the throughput of the system can match the incoming RPS, as well as, not violate SLO. To find the optimal point we search the configurations along each curve and plot a yellow point indicating the highest RPS (Figure 5a) or lowest power (Figure 5b) on the curve, respectively.

Intuitively, by scaling down frequency we see consistent drops in RPS, indicated by going down the contours. However, with resource scaling, the RPS remains stable to various degrees. This is because inference workloads have been shown to under-utilize the GPU. ***This allows us to restrict compute resources to reduce power but without having any effect on the server's throughput.*** Specifically, for `albert`, `alexnet`, and `vgg19`, the optimal configuration for a specific RPS, is at the knee point of the curve where resource scaling starts to affect RPS drastically. However, at around 15-10 active CUs, resource scaling becomes so extreme that the RPS is limited at all frequency level, indicated by the contours becoming vertical. The contour lines for `shufflenet` and `squeeznet` are not as defined as they under-utilize the GPU to such a degree that frequency and resource scaling have little effect on RPS.

CU-level vs SE-level power gating Lastly, we compare the optimal configurations based on either CU-level power gating (Figure 6a or SE-level power gating (Figure 6b). Each figure shows the geomean of all workloads RPS and power. Here

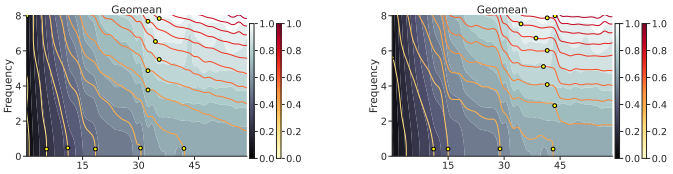


(a) Normalized Power Curves (Color bar) with yellow points indicating highest throughput for a given power level.



(b) Normalized RPS Curves (Color bar) with yellow points indicating lowest power consumption for a given RPS level.

Fig. 5: Characterizing the effect of frequency and resource scaling ***with modeled CU/SE power gating***. **Green heat map** represents the achieved latency, with **black boxes** indicating SLO violation. (a) Top figure, shows the measured power adjusted for CU-granularity power gating, with **yellow points** indicating the configuration that achieves the highest throughput for a given RPS contour level. (b) Bottom figure, shows the achieved RPS as contours, with **yellow points** indicating the configuration that achieves the lowest power consumption for that contour level.



(a) CU Gated

(b) SE Gated

Fig. 6: Geomean of workloads showing RPS (blue, filled contours) and power (red contour lines), for both CU (a) and SE (b) level power gating. CU-level gating allows more aggressive resource scaling.

RPS is shown as the (background) blue, filled contours and power has the red contour lines on top. Again, each power line is marked with the configuration that achieves the highest RPS. On average the optimal amount of resources for a per CU power-gated GPU is just over the 30 CU mark. While, for an SE power-gated GPU, it is just below 45 CUs, as the SE will not be gated unless all CUs are inactive. In general, CU-level gating provides more aggressive resource scaling opportunities for power savings. While the max RPS for the lowest power levels are at the lowest frequency scale, this is also typically where there start to be SLO violations, making these configurations invalid.

B. CoFRIS Implementation

The design objective of CoFRIS is to minimize power of a GPU while maintaining SLO for a variable incoming request rate. Figure 7 presents an overview of CoFRIS. CoFRIS consists of offline profiled frequency-resource response curves for a given inference model, along with an online runtime that dynamically determines the frequency/resource scaling configuration given a model's incoming RPS rate.

We utilize the observations stated in subsection III-A to configure the GPUs frequency and amount of available resources. As we saw previously, each inference model has drastically unique tolerances to frequency scaling, resource scaling, and SLO tolerance. Due to this, we profile each model's sensitivity

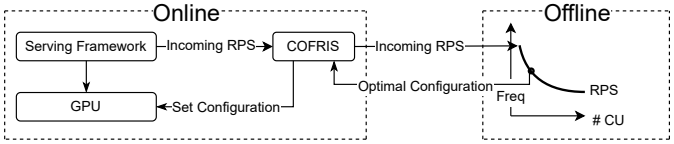


Fig. 7: Overview of CoFRIS

offline similar to many prior works on spatially partitioned GPU-based inference servers [5], [8], [20]. First, we determine the maximum RPS and baseline tail latency based on running the model at the max frequency and CUs. This gives us our model's RPS range for our system. Next, at each RPS step from 0 to max, the optimal frequency/resource configuration is determined to be the one that minimizes GPU power, while having no SLO violations. This process is seen in Figure 7 inside the Offline box.

This frequency/resource response curve is then stored in a table in the serving framework runtime. Then, during runtime, CoFRIS takes incoming RPS information from the serving framework (i.e. TorchServe, Trition, TensorflowServing, etc.) and looks up the optimal configuration from the table and sets the frequency and available compute resources. Polling is done every 0.5 seconds, and uses user-level APIs (either through ROCm SMI or CU Masking APIs) to set the configuration.

IV. Evaluation

A. Evaluation Methodology

We evaluated CoFRIS on a server featuring an AMD MI50 GPU, 2 AMD EPYC 7302 16-Core Processor, 512 GB RAM, Ubuntu 18.04 LTS with kernel 5.4.0. The AMD MI50 GPU contains 60 Compute Units across 4 Shader Engines. The server runs the AMD ROCm 5.2 runtime stack. ***While CU power gating does exist on the hardware, but is not exposed to consumers [3], [26], [27]., we evaluate the benefits of CU power gating as previously detailed in Section II-B.***

Inference server. For our evaluation, we built our own custom inference server framework as most existing inference servers, such as TensorRT, are designed for Nvidia-based

TABLE I: Inference workload used and 95% tail latency (ms).

Model	95% lat. (ms)
albert [23]	27
alexnet [22]	91
densenet201 [15]	72
resnet152 [13]	11
resnext101 [35]	154
shufflenet [24]	8
squeezenet [16]	8
vgg19 [31]	81

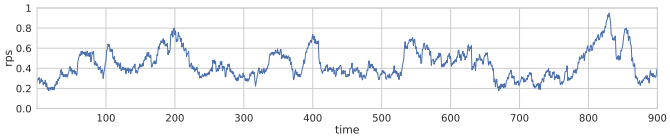


Fig. 8: Client Request Trace. Derived from Facebook's SWIM Dataset [32]

GPU systems and tightly integrate Nvidia-specific features. Our inference server consists of (1) an *Inference Front-end*, a multi-threaded process responsible for accepting asynchronous gRPC requests from clients and sending back the inference result (response), (2) *Request/Response Queues*, where queues are shared memory segments for storing request's (response's) data to be served (sent to the client)., and (3) *Workers*, where each worker is an instance of a ML framework (such as PyTorch, Tensorflow, etc.) that services the inference request.

Inference model Workloads. The inference models evaluated are listed in Table I. For this work, we fix our request's batch size to 1, as this represents scenarios where servers require very low latency. This will also allow us to show the impact of scaling on the GPU without any interaction with dynamic batch sizing. However, previous works have seen improvement in coordinating the batch size and DVFS [28] and there can be future opportunities to coordinate all three. We used Facebook's SWIM dataset [32] as the basis of our client request generator. We normalized the trace to be fifteen minutes long and each workload's max RPS being a load of 1 as shown in Figure 8.

Power management policies. To evaluate CoFRIS, we compare against six scaling policies as follows:

`Baseline`: No frequency or resource scaling is used. This means the GPU is consistently running at max frequency with all available CUs.

`FS`: Frequency Scaling using the max resources.

`RS-SE Gated`: Resource Scaling with power-gating at the SE-granularity.

`RS-CU Gated`: Resource Scaling with power-gating at the CU-granularity.

`COFRIS-SE Gated`: Coordinated Frequency and Resource Scaling with power-gating at the SE granularity.

`COFRIS-CU Gated`: Coordinated Frequency and Resource Scaling with power-gating at the CU granularity.

1) Results: **GPU Power:** Figure 9 plots the average power of the GPU normalized to our baseline policy. On average `RS-SE Gated` and `FS` show similar power reduction.

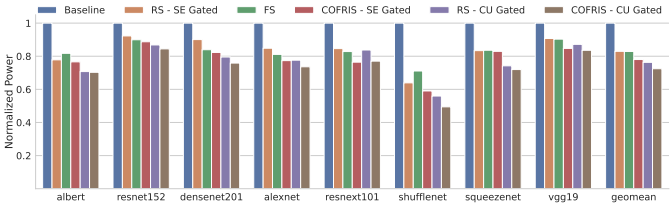


Fig. 9: Average Power for each power management policy.

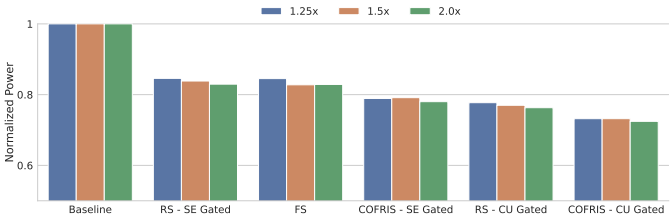


Fig. 10: Geomean of average power for each policy, with varying SLO constraints.

`RS-CU Gated` demonstrates the potential savings of having a finer power gating granularity which performs better than `RS-SE Gated` except for `alexnet`, `resnext101`, and `vgg19`. However, individual models show varying amounts of sensitivity to power management policies, e.g. `RS-SE Gated` saves more power for `albert`, `shufflenet`, and `squeezenet`, therefore, there is no clear winner. This demonstrates that resource scaling can potentially provide more savings than frequency scaling.

One important point to note is the granularity between frequency vs resource steps. With our GPU we can control each CU individually but are limited to only 9 frequency steps that are not uniform in distance (as illustrated in Figure 3). This may account for `RS-CU Gated` outperforming `FS` due to the limitations of frequency steps. It may be possible if there were more steps at lower frequencies we would be able to see more improvement from frequency scaling. Future exploration in under-clocking along with voltage control may lead to more opportunities. In any regard, we show that coordinating frequency and resource scaling has the potential to lead to the most power savings.

Coordinating both frequency and resource scaling with `COFRIS-SE Gated` saves 6% more power over `FS`. This indicates that neither resource nor frequency scaling alone can exploit the latency slack to its fullest extent and by coordinating both we can extract more power savings opportunities. In total, `COFRIS-CU Gated` outperforms all other policies With 28% average power decrease over baseline, 13% improvement over `FS`, and 5% over the next best which is `RS-CU Gated`. ***This demonstrates the energy savings potential for enabling CU/SE power gating for consumer GPU devices.***

Sensitivity to SLO: Next we explore how frequency/resource scaling can be affected by various levels of latency constraint. Figure 10 displays that power savings can still be achieved even when the latency slack is tightened. While we do observe the trend that having a larger slack means a larger

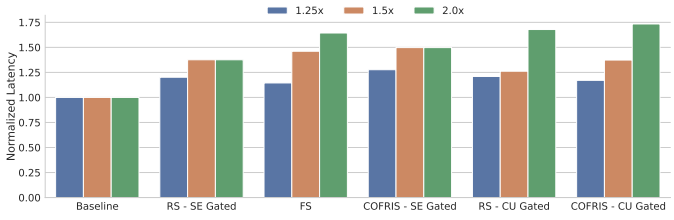


Fig. 11: 95th percentile tail latency achieved for each policy. Latency slack is used up while maintaining SLO.

power reduction, all policies are able to achieve almost the same level of power savings (within 2%) with a slack of 1.25x and 1.5x compared with 2.0x.

Finally, to evaluate if we are meeting SLOs, we plot the geomean of 95th percentile tail latency achieved for each policy in Figure 11. In all scenarios, we meet the SLO. `RS-SE Gated` and `COFRIS-SE Gated` flatten out after 1.5x. This is due to the fact that the SE-gated policies most often choose the resource configuration of 45 CUs, as choosing any less doesn't save any power but does lower the RPS it is able to handle even if it is within SLO. `FS` and CU-gated policies are able to achieve tail latency closer to the SLO target and minimize latency slack, however at most only reaches 1.75x. This indicates that there is possibly more latency slack opportunity that can be pushed closer to the SLO for power savings opportunity.

V. Related Works and Discussion

Prior works have used frequency scaling [19], deep sleep states [7], or coordination of both [6] to reduce power for latency-critical services on CPUs by closing the latency slack. GPUs frequency and power states have also been leveraged to optimize the performance and power efficiency of individual kernels [25]. Resource scaling has been used explicitly for GPUs as they are parallel processors with many cores [29], [34].

Specifically for GPU inference servers, frequency scaling has been combined with dynamic batching [28]. For our work, we specifically propose coordinating frequency and resource scaling for latency-critical and dynamic throughput inference servers. While dynamic batching has been shown to improve utilization it may not always be possible to increase the batch size, as it leads to significant increases in tail latency and it might lead to SLO objectives.

Prior works have also looked at resource scaling within inference servers. InferLine [9] provisions and scales at the GPU level in a *GPU cluster* while maintaining SLO, while our work focuses on resource scaling within a *single GPU*. AutoInfer [4] coordinates a resource scaling and batch size to improve total GPU utilization, however, they do not try to minimize GPU power consumption for their configurations. Our work differs from prior works as we coordinate resource and frequency scaling within a single GPU to minimize GPU power, while showcasing the potential impact of CU or SE level power gating.

A. Implications of Fine-grain Power Gating in GPUs:

Prior works have shown that power gating at fine granulates comes with extra area overhead costs along with diminishing returns in energy savings. [10] While we do not analyze area overhead in this work, past works have shown that AMD GPUs have some form of per CU gating, but was only achieved with internal tools [26], [27]. ***This work aims to showcase the potential savings with either fine-grain per CU Gating or per SE gating and hope to motivate industry to bring these internal tools to external researchers or these features into commercial products.***

Although we modeled the potential power savings of CU/SE power gating, we believe our findings can be generalized to other GPU architectures as many architectures have similar architectural organizations. We believe this work can be generalized to any GPUs from AMD or Nvidia. Nvidia uses similar structures in their GPUs, they call CU a Streaming Multiprocessor (SM) and the SE is called a Graphics Processing Cluster (GPC). CPUs have long benefited from fine-grain power gating of compute units through C-states, and we believe our results motivate the need for similar fine-grain power gating states for GPUs to unlock further energy efficiency potentials.

VI. Conclusion

Latency slack in inference servers leaves a large gap between the average latency and service level objectives. This slack can be exploited by scaling the GPUs frequency and resources to reduce total GPU power usage without violating SLO. While DVFS has been previously explored to bridge the slack gap, our work shows that it typically is not able to push the average latency far enough, leaving untapped power savings. We propose CoFRIS which coordinates frequency and resource scaling for GPU inference servers. ***During our initial exploration, we found that current AMD GPUs do not automatically initiate power gating, and thus evaluate using modeled CU / SE power gating. We hope this work motivates the importance of enabling programmer control over power gating.*** We show that SE-level power gating can be effective, but finer-grain CU-level power gating proves to be the most efficient. In total, CoFRIS with CU-level power gating lowers power by 28% over baseline, a 13% improvement over *FS*, and 5% over isolated CU-power gated resource scaling ***with modeled CU/SE power gating***.

Acknowledgement

This work is partly supported by National Science Foundation under grants CNS-1955650 and CNS-2047521. We would also like to thank the anonymous reviewers for their invaluable comments and suggestions.

References

[1] AMD, "rocm_smi_lib." [Online]. Available: https://github.com/RadeonOpenCompute/rocm_smi_lib
[2] AMD, "Stream management hip api." [Online]. Available: https://docs.amd.com/bundle/HIP-API-Guide-v5.4.1/page/a00183.htmlf
[3] AMD, "Changing number of compute units issue #5 radeonopencompute/roc-smi," 2017. [Online]. Available: https://github.com/RadeonOpenCompute/ROC-smi/issues/5

[4] B. Cai, Q. Guo, and X. Dong, "Autoinfer: Self-driving management for resource-efficient, slo-aware machine learning inference in gpu clusters," *IEEE Internet of Things Journal*, vol. 10, no. 7, pp. 6271–6285, 2022.

[5] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Multi-model machine learning inference serving with gpu spatial partitioning," *arXiv preprint arXiv:2109.01611*, 2021.

[6] C.-H. Chou, L. N. Bhuyan, and D. Wong, "μDPM: Dynamic power management for the microsecond era," in *HPCA*, 2019.

[7] C.-H. Chou, D. Wong, and L. N. Bhuyan, "Dynsleep: Fine-grained power management for a latency-critical data center application," in *ISLPED*, 2016.

[8] M. Chow, A. Jahanshahi, and D. Wong, "KRISP: Enabling kernel-wise right-sizing for spatial partitioned gpu inference servers," in *HPCA*. IEEE, 2023.

[9] D. Crankshaw, G.-E. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, and A. Tumanov, "Inferline: latency-aware provisioning and scaling for prediction serving pipelines," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 477–491.

[10] K. Dev, S. Reda, I. Paul, W. Huang, and W. Burleson, "Workload-aware power gating design and run-time management for massively parallel gpgpus," in *ISVLSI*. IEEE, 2016.

[11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *international symposium on Computer architecture*, 2011.

[12] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *OSDI*, 2020.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[14] Y. Hu, R. Ghosh, and R. Govindan, "Scrooge: A cost-effective deep learning inference system," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *conference on computer vision and pattern recognition*, 2017.

[16] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[17] A. Jahanshahi, H. Z. Sabzi, C. Lau, and D. Wong, "GPU-NEST: Characterizing energy efficiency of multi-gpu inference servers," *IEEE Computer Architecture Letters*.

[18] P. Jain, X. Mo, A. Jain, H. Subbaraj, R. S. Durrani, A. Tumanov, J. Gonzalez, and I. Stoica, "Dynamic space-time scheduling for gpu inference," *arXiv preprint arXiv*, 2018.

[19] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *International Symposium on Microarchitecture*, 2015.

[20] Y. Kim, Y. Choi, and M. Rhu, "PARIS and ELSA: an elastic scheduling algorithm for reconfigurable multi-gpu inference servers," in *DAC*, 2022.

[21] J. Kosaian, A. Phanishayee, M. Philipose, D. Dey, and R. Vinayak, "Boosting the throughput and accelerator utilization of specialized cnn inference beyond increasing batch size," in *ICML*, 2021.

[22] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[23] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[24] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *ECCV*, 2018.

[25] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi, "Dynamic gpgpu power management using adaptive model predictive control," in *HPCA*, 2017.

[26] A. Majumdar, G. Wu, K. Dev, J. L. Greathouse, I. Paul, W. Huang, A.-K. Venugopal, L. Piga, C. Freitag, and S. Puthoor, "A taxonomy of gpgpu performance scaling," in *IISWC*, 2015.

[27] A. McLaughlin, I. Paul, J. L. Greathouse, S. Manne, and S. Yalamanchili, "A power characterization and management of gpu graph traversal," in *ASBD*, 2014.

[28] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[29] I. Paul, W. Huang, M. Arora, and S. Yalamanchili, "Harmonia: Balancing compute and memory power in high-performance GPUs," *ISCA*, 2015.

[30] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *USENIX ATC 21*, 2021.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] SWIMProjectUCB, "Statistical workload injector for mapreduce (swim)," 2016. [Online]. Available: https://github.com/SWIMProjectUCB/SWIM/wiki

[33] Q. Weng, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *NSDI*, 2022.

[34] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "Antman: Dynamic scaling on gpu clusters for deep learning." in *OSDI*, 2020.

[35] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.