BatchSizer: Power-Performance Trade-off for DNN Inference

Seyed Morteza Nabavinejad
nabavinejad@ipm.ir
Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran

Sherief Reda
sherief_reda@brown.edu
Brown University
Providence, RI

Masoumeh Ebrahimi
mebr@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

GPU accelerators can deliver significant improvement for DNN processing; however, their performance is limited by internal and external parameters. A well-known parameter that restricts the performance of various computing platforms in real-world setups, including GPU accelerators, is the power cap imposed usually by an external power controller. A common approach to meet the power cap constraint is using the Dynamic Voltage Frequency Scaling (DVFS) technique. However, the functionally of this technique is limited and platform-dependent. To improve the performance of DNN inference on GPU accelerators, we propose a new control knob, which is the size of input batches fed to the GPU accelerator in DNN inference applications. After evaluating the impact of this control knob on power consumption and performance of GPU accelerators and DNN inference applications, we introduce the design and implementation of a fast and lightweight runtime system, called BatchSizer. This runtime system leverages the new control knob for managing the power consumption of GPU accelerators in the presence of the power cap. Conducting several experiments using a modern GPU and several DNN models and input datasets, we show that our BatchSizer can significantly surpass the conventional DVFS technique regarding performance (up to 29%), while successfully meeting the power cap.

CCS CONCEPTS

• **Computer systems organization → Single instruction, multiple data**; *Neural networks.*

KEYWORDS

batch size, inference, deep neural networks, power cap

ACM Reference Format:
Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2021. BatchSizer: Power-Performance Trade-off for DNN Inference. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431535

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431535

1 INTRODUCTION

With the proliferation of DNNs in various applications such as speech recognition [13], computer vision [15], and natural language processing [14], a large body of research has focused on accelerating training and inference phases of DNNs for different platforms such as GPUs, FPGAs, and ASICs. The GPU accelerator is a popular option for improving the performance of DNNs due to its programmability and scalability features and the proven promising results. While in theory, a GPU accelerator can exploit its maximum power capacity when executing applications, in a real-world setup, the available power budget is capped by an external agent. The reason is that different components of a computing platform such as CPU, memory modules, storage, network interface, and hardware accelerators (e.g., GPU accelerator) usually share the power source. Consuming more power by a component than its allocated amount means power shortage in other components. The power cap can severely degrade the performance of DNNs on GPU accelerators, and hence, one should employ proper techniques and methods to address this challenge.

The common practice to maintain the power usage within the power cap in many processors such as CPUs and GPUs is applying dynamic voltage frequency scaling (DVFS) on that processor. A large body of research has studied power management in GPU accelerators and have proposed various approaches based on DVFS [3, 8, 11]. While many GPU vendors have developed user-friendly interfaces for DVFS management (e.g., nvidia-smi by Nvidia [1]), the application of those interfaces is restricted due to several obstacles. First, the DVFS levels accessible on different GPUs are limited to the ones designed by the GPU vendor, and the end-user cannot set the DVFS to a level beyond the available ones. Consequently, the power consumption of the GPU accelerator is also limited to a set of certain values. Therefore, it is hard to efficiently manage the power consumption to meet the power cap while maximizing the accelerator performance. Second, increasing the DVFS level does not certainly lead to proportional increases in performance. An application (e.g., DNNs) that is running on GPU should also be able to fully utilize the available resources such as streaming multiprocessors (SMs). Otherwise, DVFS increases the working frequency of GPU streaming multiprocessors without any usage.

To address the aforementioned shortages of the conventional DVFS technique in GPU-based DNN inference accelerators, we introduce a new control knob which is the size of input batches during DNN inference. We show that the size of a batch has a direct impact on resource utilization, the power consumption of the GPU accelerator, as well as the performance of DNN inference. Leveraging this new control knob, we design and implement a lightweight runtime system called *BatchSizer*. It can determine the batch size for each time epoch considering the power cap of the

GPU accelerator, such that the throughput of the DNN inference is maximized. *BatchSizer* is based on binary search, and hence, can select the proper batch size in a fraction of time.

We conduct various experiments to evaluate the efficacy of *BatchSizer* and compare its performance against the DVFS technique. We employ a P40 GPU as the accelerator, several well-studied image classification DNNs as workloads, and two popular image datasets as the input of DNN inference. The experimental results show that *BatchSizer* can maintain the power cap while improving the throughput by up to 29% compared with another approach that leverages DVFS.

The main contributions of the paper are as follows:

- We study the impact of batch size on resource utilization of the GPU accelerator and observe how different batch sizes affect power consumption and performance of DNN inference.
- We implement dynamic batch sizing for image classification DNNs that enables us to change the batch size of DNN inference on the fly, with least possible overhead and without any interrupt.
- We design and implement a lightweight runtime system called *BatchSizer* that can find the proper batch size in a few steps using binary search such that the power consumption is less than the power cap, while the throughput is maximized.
- Conducting an extensive set of experiments, we show the effectiveness of BatchSizer compared to an approach that leverages traditional DVFS technique.

The rest of the paper is organized as follows: we summarize the related works in Section 2 and discuss the motivation behind our study in Section 3. Our proposed approach is introduced in Section 4. We evaluate our approach and present results of experiments in Section 5. Finally, we conclude the paper in Section 6.

2 PREVIOUS WORK

Employing Dynamic Voltage Frequency Scaling (DVFS) to improve the power or performance of GPU accelerators has been explored in a large body of research [3, 8, 11, 17, 18]. Komoda *et al.* [9] proposed a power capping technique for CPU-GPU systems that considers DVFS and task mapping simultaneously. They tended to set the frequency of both CPU and GPU according to the task mapped to each of them. Therefore they model the performance and power consumption of the CPU-GPU system considering the task mapping and DVFS level to prevent power cap violation or load imbalance. To enhance the performance per watt of GPU accelerators, Jiao *et al.* [8] leveraged the cutting-edge GPUs' ability to host several concurrent kernels, and applied core and memory frequency scaling. Guerreiro *et al.* [3] classified the effect of DVFS on the performance and power of GPU applications to obtain representative models. Based on the models, they estimated the impact of various DVFS settings on the performance and power of applications to set the DVFS of the GPU accelerator on the most proper level. GreenMM [18] aimed to reduce the energy consumption of matrix multiplication, a prevailing operation in many DNNs, including image classification ones, in GPU accelerators. They employed GPU undervolting without reducing the frequency to decrease the power consumption as much as possible. Since aggressive undervolting might result in an increased fault rate, GreenMM introduces Algorithm-Based Fault Tolerance (ABFT) to mitigate this challenge. Tang *et al.* [17] conducted an extensive set of experiments to understand the impact of DVFS on performance and energy consumption of several DNNs. To this end, they executed four DNNs on three GPU accelerators. PIT [11] is another approach that employs DVFS along with reduced-precision instructions supported by new GPUs to manage the power consumption of DNN inference on GPU accelerators. It first deploys the reduced-precision model of DNN on the GPU, and if needed, it adjusts the GPU frequency with its dedicated procedure. That procedure starts from the lowest DVFS level and increases it as much as possible to improve the performance, while meeting the power cap.

3 MOTIVATION

It this section, we aim to show the impact of the batch size on power and performance of DNN inference. We employ three DNNs, namely Inception [16], ResNet [4], and MobileNet [6] and two image datasets as input with 10,000 images, one from the ImageNet dataset [12], and the other from Caltech256 dataset [2]. We deploy the three DNNs on an Nvidia P40 GPU in sequence, and repeat that several times to process the two datasets. In each iteration, we double the batch size and monitor the throughput (image per second), as well as the power consumption and resource utilization of the GPU (later in Section 5, we discuss the experimental setup in more detail).

The throughput and maximum power consumption are shown in Fig. 1, which reveal the strong relationship between the batch size and power and throughput. With increasing the batch size, both throughput and maximum power consumption also increase. However, the effect of batch size depends on the DNN and dataset. For example, when processing the Caltech256 dataset using MobileNet, we observe significant throughput improvement as the batch size increases, while the improvement is negligible when processing the ImageNet dataset with the same DNN. Furthermore, we observe

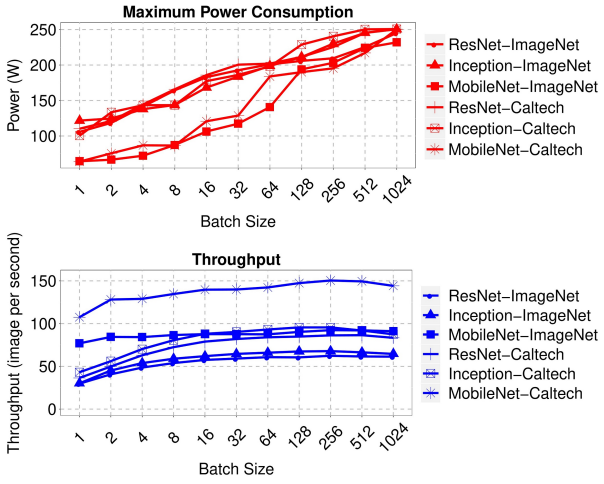

Figure 1: Impact of the batch size on throughput and power consumption of DNN inference.

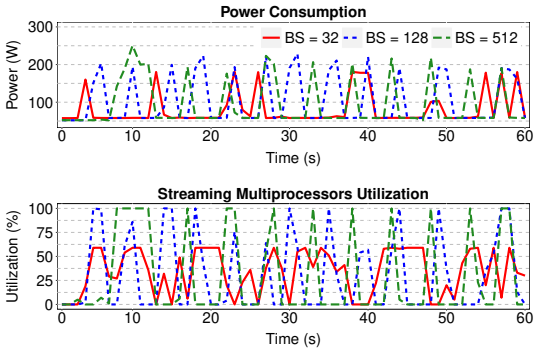

Figure 2: Relationship between resource utilization of GPU, its power consumption, and the maximum throughput achievable by increasing the batch size.

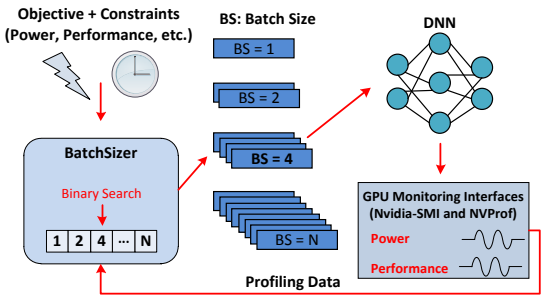

Figure 3: Overall flow of BatchSizer

that after the batch size 256, the throughput does not increase (or even decreases) while the maximum power consumption continues to rise.

To better understand the cause of such behavior, in Fig. 2 we depict the power consumption and Streaming Mutliprocessors (SMs) utilization over time for one of the DNNs (Inception with the Caltech256 dataset) under three different batch sizes: 32, 128, and 512. After BS = 128, the maximum SMs utilization approaches 100%, and it is exactly 100% for BS = 512. Therefore after BS = 256, increasing the batch size only leads to increased power consumption as the utilization of other resources (e.g., memory) increases. On the other hand, the throughput does not improve (or even degrades) because of contention among active GPU threads over shared resources such as memory bandwidth. We consider this observation, along with other ones, when designing the BatchSizer. Finally, we observed that both power consumption and SMs utilization are significantly fluctuating. As presented in Fig. 2, when the input batch is being prepared for execution, the power consumption is low and SMs utilization is zero because no computation is happening. But when input execution starts, spikes happen on the SMs utilization, and consequently, consumed power. These spikes determine the maximum power consumption and SMs utilization.

4 METHODOLOGY

4.1 Problem Statement

The problem that we aim to address in this work is as follows: A DNN inference application i is deployed on a GPU accelerator with a default batch size BS. A power cap (P_{cap}) is applied to GPU accelerator by an external controller, and the power consumption of GPU should be less than that. Both throughput and power consumption of DNN_i are a function of the batch size, in addition to other parameters such as input dataset, temperature of GPU, etc. The objective function is to maximize the throughput over the course of time while meeting the power cap. The T parameter shows the period of time that the job is active and running.

$$Maximize \quad \frac{1}{T}\sum_{t=1}^{T} Throughput_i^t \tag{1}$$

s.t.

$$Power_i^t \leq P_{cap} \tag{2}$$

4.2 BatchSizer

A key prerequisite to have a runtime system that can effectively decide on the batch size during the execution of DNNs is implementing dynamic batch sizing in the applications. Currently, the batch size can be determined only when the application is submitted to the GPU accelerator. Once the application is deployed and started the execution, the batch size cannot be changed any longer. The conventional approach to change the batch size is to terminate the running instance and launch a new one with a different batch size, which imposes significant delay or reduces the average throughput. To mitigate this challenge, we implement dynamic batch sizing by modifying a few lines of code compared with conventional static batch size. The changes are straightforward and do not degrade the programmability of the DNNs. Besides, it imposes almost no notable overhead on latency or throughput. Dynamic batch sizing enables us to change the batch size on the fly without any interrupt in the flow of DNN inference. Having dynamic batch sizing implemented, we can proceed to the design and implementation of our runtime system, *BatchSizer*.

The design objective of *BatchSizer* is to maximize the average throughput while considering the power cap. To achieve this objective, *BatchSizer* dynamically changes the batch size over the course of time. In the design of *BatchSizer* we take into account the observations presented in Section 3. The first key observation is that both maximum power consumption and throughput increase with the batch size. Therefore we can assume that those parameters are sorted in ascending order with respect to the batch size. Having them sorted, *BatchSizer* can employ a pseudo binary search approach to efficiently search the state space in a few steps and find the most suitable batch size. Since the time complexity of the binary search is $O(\log n)$, the time overhead of *BatchSizer* is negligible.

The second observation is the spiking nature of the power consumption (see Fig. 2), which is used in the design of *BatchSizer*. Considering the maximum power consumption, which occurs during the spikes, is essential to make sure that the power consumption does not surpass the power cap. Thereby, after changing the batch size, *BatchSizer* waits for a few batches to be processed to profile the power consumption of a few number of spikes. The third observation used in the design of *BatchSizer* is the throughput saturation after a specific batch size (see Fig. 1). As we mentioned in Section

3, after BS = 256, we see no improvement in throughput. *BatchSizer* skips those batch sizes and only considers the ones that lead to throughput improvement. In the current version, *BatchSizer* uses offline profiling information to detect the maximum beneficial batch size. However, it can be easily modified to detect them online, but with a certain overhead. In the online approach, after selecting a batch size and monitoring its throughput, *BatchSizer* can evaluate the throughput of an immediate smaller batch size to ensure that the current batch size offers a better throughput than a smaller one. Otherwise, it skips the current batch size and selects the smaller one.

Combining the three aforementioned observations, the overall flow of *BatchSizer* can be described as follow, also presented in Algorithm 1 and Fig. 3. *BatchSizer* starts with a default batch size (1 in our experiments). After processing a few input batches (10 in our experiments) and monitoring the power consumption, it compares the profiled power data with the power cap. If the maximum power consumption is equal or less than a certain coefficient of the power cap ($\alpha \times P_{cap}$), then the current batch size is good, and no further action is needed. We have set $\alpha = 0.8$ in the experiments. We consider a lower margin of the power cap to avoid excessive batch size changes that might lead to an enormous number of power cap violation instances. If the maximum power consumption is greater than the power cap, *BatchSizer* sets the batch size as the median of the lowest possible batch size and the current batch size. If the current batch size is the smallest one, no further batch size reduction is possible, and hence, the power cap cannot be met in the current state. If the maximum power consumption is less than $\alpha \times P_{cap}$, it means there is room to improve throughput by increasing the batch size. Therefore, *BatchSizer* sets the batch size to be equal to the median of the current batch size and the largest possible batch size. If the current batch size is the largest one, then no further throughput improvement is possible through batch sizing. Fig. 4 shows how BatchSizer works with the help of an illustrative example.

Since *BatchSizer* continues processing the input data even when it is searching for a proper batch size, it does not impose any time or resource overhead on the system. However, the throughput may slightly degrade when the *BatchSizer* is searching, or the power cap might be violated for a short period of time. The other design feature of *BatchSizer* is the continuous batch size adjustment. Upon finding a suitable batch size it does not stop, but continues monitoring the power consumption. It restarts batch size adjustment again if it detects power cap violation or throughput improvement opportunities that might happen due to changes in the power cap or power consumption of the DNN. The power cap might be changed by external power controller and the power consumption of DNN can be affected by parameters such as variation in input data.

Algorithm 1 BatchSizer

Input: P_{cap}; SB(1:N): Set of available batch sizes in an ascending order
minBS = 1; maxBS = N; initialBS = 1; currentBS = 1; BS = SB(initailBS); PowerReading = []
while True **do**
  PowerReadng.append(monitorPower(inference(BS)))
  if $\alpha \times P_{cap} \leq$ max(PowerReading) $\leq P_{cap}$ **then**
    Continue with BS
  if max(PowerReading) $< \alpha \times P_{cap}$ **then**
    minBS = currentBS
    currentBS = ceil($\frac{minBS+maxBS}{2}$)
    BS = SB(currentBS)
  if max(PowerReading) $> P_{cap}$ **then**
    if currentBS = 1 **then**
      Further BS reduction is not possible.
    if currentBS = minBS **then**
      maxBS= currentBS, minBS = 1,
      currentBS = floor($\frac{minBS+maxBS}{2}$)
      BS = SB(currentBS)
    else
      maxBS = currentBS
      currentBS = floor($\frac{minBS+maxBS}{2}$)
      BS = SB(currentBS)

5 EVALUATION

5.1 Evaluation Setup

We employ a dual-socket Xeon server equipped with two E5-2680 v4 Xeon chips each with 28 cores running at 2.4 GHz and 128GB of DDR4 memory. Ubuntu 16.04 with kernel 4.4 is installed on the server with CUDA 11.0 and TensorFlow 1.15. A Tesla P40 GPU Accelerator is attached to the server that is based on Nvidia Pascal architecture and has 3840 CUDA cores and 24GB GDDR5 memory, and its maximum power limit is 250W. We employ 10 DVFS levels for GPU in our experiments: 544 MHz, 632 MHz, 734 MHz, 835 MHz, 949 MHz, 1063 MHz, 1189 MHz, 1303 MHz, 1430 MHz, and 1531 MHz. Please note we can only control the frequency of GPU. The GPU driver automatically adjusts the voltage according to the selected frequency. To measure the power consumption of GPU, we employ nvidia-smi tool which uses the embedded sensors of GPU hardware for reading power consumption over the time. We choose ten DNNs with different characteristics such as size and computational complexity to show the applicability of *BatchSizer* on a wide variety of DNNs. The specifications of the DNNs are presented in Table 1. We have two image datasets, one from ImageNet [12] which is a popular dataset that is widely used in other works, and the other one is Caltech 256 [2] which is collected by researchers from the California Institute of Technology. The workload used in the experiments consists of 20 jobs shown in Table 2. Each DNN model from Table 1 has appeared two times in the workload, once with input from ImageNet, and once with input from Caltech 256. The power cap for each job is a number between 50 W (the minimum power when loading a model on GPU) and 250 W (the maximum power capacity of GPU). These numbers are selected randomly, but we have tried to cover a wide range of power caps, from tight caps to relaxed ones.

The batch size set that *BatchSizer* selects from is as follows: {1, 2, 4, 8, 16, 32, 64, 128, 256}. While we have used them in the experiments, any other set with an arbitrary size can also be used. Since the time complexity of *BatchSizer* is *O (log n)*, it can handle large sets and find a suitable batch size with the least number of tries and within a reasonable time.

Previous works such as PIT [11] usually consider a simple routine for changing the DVFS (starting from the least amount and increasing it step by step until reaching the power cap). To have a more sophisticated approach for comparison, similar to *BatchSizer*, we employ a pseudo binary search approach in all experiments to find a proper DVFS level. To have a fair comparison, we select BS = 16 (middle of the batch size set) for this DVFS-based algorithm.

Table 1: Description of DNNs used in the experiments

Name	Abbr	Frozen Graph Size (MBs)	Computational Complexity (MFLOPs)	Reference
Inception V1	IncV1	26.06	13.22	[15]
Inception V2	IncV2	43.92	22.34	[7]
Inception V3	IncV3	93.42	54.25	[16]
Mobilenet-V1-1	MobV1	16.69	8.42	[6]
Mobilenet-V2-1	MobV2	13.75	6.94	[5]
NASNET-Mobile	NASNET	21.37	10.51	[19]
PNASNET-Mobile	PNASNET	20.37	10.06	[10]
ResNet-V2-50	Res50	100.21	51.01	[4]
ResNet-V2-101	Res101	174.82	88.89	[4]
ResNet-V2-152	Res152	236.32	120.08	[4]

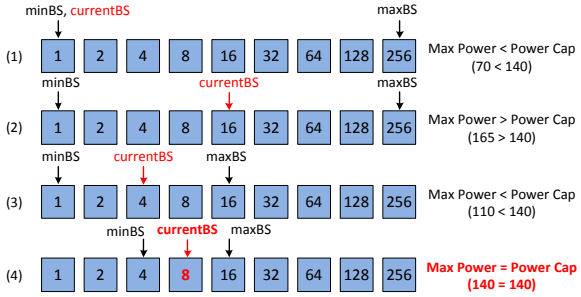

Figure 4: An illustrative example to show the workflow of *BatchSizer*

Table 2: Specification of Jobs Used in the Experiments

	DNN	Dataset	Power Cap (W)		DNN	Dataset	Power Cap (W)
1	IncV1	ImageNet	191	11	IncV1	Caltech	125
2	IncV2	ImageNet	123	12	IncV2	Caltech	192
3	IncV3	ImageNet	228	13	IncV3	Caltech	165
4	MobV1	ImageNet	195	14	MobV1	Caltech	98
5	MobV2	ImageNet	84	15	MobV2	Caltech	208
6	NASNET	ImageNet	159	16	NASNET	Caltech	135
7	PNASNET	ImageNet	246	17	PNASNET	Caltech	136
8	Res50	ImageNet	143	18	Res50	Caltech	198
9	Res101	ImageNet	167	19	Res101	Caltech	236
10	Res152	ImageNet	218	20	Res152	Caltech	188

Selecting a small batch size (e.g., 1) leads to poor throughput, while a large one (e.g., 256) causes significant power cap violations.

5.2 Experimental Results

First, we present the throughput results in Fig. 5. *BatchSizer* can improve the throughput in most of the jobs (up to 29% in job 19) while yielding slightly lower throughput than DVFS in some others (up to 8% lower in job 5). To understand the reason behind these improvements or slight reductions, we show the average batch size selected by *BatchSizer* for each job in Fig. 6. For the jobs that the average batch size is close to the DVFS batch size (i.e., 16), the *BatchSizer* either has lower throughput or slight throughput improvement, compared with DVFS. The reason for such small batch sizes selected by *BatchSizer* is the tight power cap of those jobs (Note that the power cap's tightness is relative to the DNN model. The same power cap, e.g., 100 W, might be tight for a deep and complex DNN such as ResNet, while too relaxed for a small DNN such as MobileNet). However, when the power cap is relaxed enough, *BatchSizer* opts for large batch sizes to fully utilize the GPU

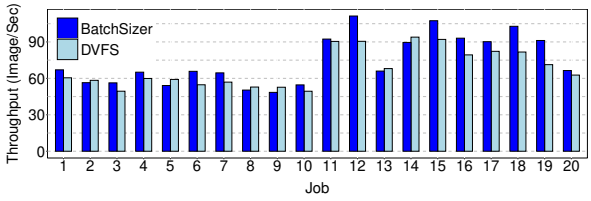

Figure 5: Comparing the throughput of *BatchSizer* against DVFS

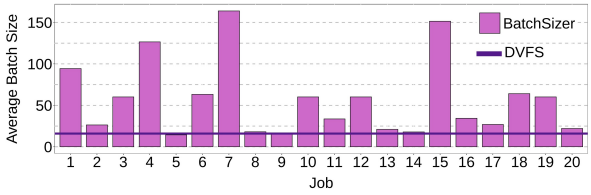

Figure 6: The average batch size employed by *BatchSizer* to process each job in comparison with constant value selected by DVFS

accelerator's computing resources, which leads to high throughput. In such cases, the DVFS cannot leverage the entire power capacity, even by the highest possible DVFS level. Hence, its throughput would be lower than *BatchSizer*.

Next, we show the power consumption pattern of both approaches for five jobs in Fig. 7. Due to the lack of space, we could not present the results for the entire workload. The results clearly show that *BatchSizer* can better exploit the power capacity by increasing resource utilization, and hence, improve the throughput. DVFS, on the other hand, cannot take full advantage of the power capacity because of its limited batch size, even by employing the highest DVFS level. Consequently, its throughput is less than *BatchSizer*. In other words, the performance of DVFS is limited to the dynamic range of power consumption achievable by the lowest and highest DVFS levels. We also observe that in some rare instances, *BatchSizer* slightly violates the power cap. While adjusting the batch size, *BatchSizer* might select the one leading to a temporary power cap violation. However, *BatchSizer* immediately modifies it and does not allow the violation to last for a long time. It is worth to mention that changing the batch size does not affect the accuracy of the results in DNNs, and hence, *BatchSizer* has no negative impact on the final accuracy of the results.

Finally, in Fig. 8, we show the dynamic behavior of *BatchSizer* regarding the batch size. The jobs plotted in this figure are the same as Fig. 7. As shown in this figure, *BatchSizer* selects a batch size to fully leverage the power capacity without violating the power cap. When there is room for more power consumption, it opts for larger batch sizes until it reaches the largest possible batch size. In the case of tight power caps, however, it tends to select smaller batch sizes. In the presence of a power cap violation, it instantly reduces the batch size to avoid further violation.

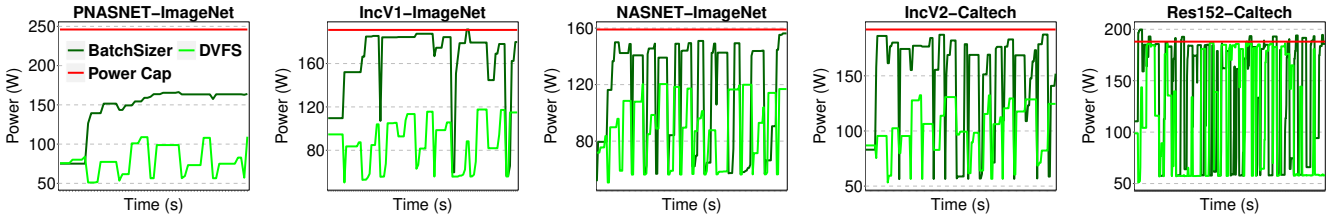

Figure 7: Showing the power behavior of *BatchSizer* and DVFS for several jobs.

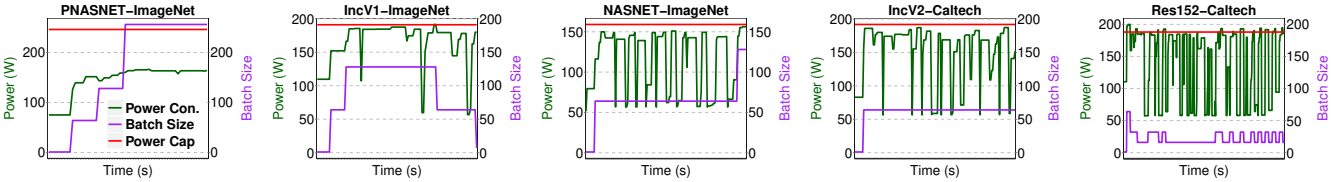

Figure 8: Illustrating the dynamic behavior of *BatchSizer* regrading the batch size selection and its impact on the power consumption.

6 CONCLUSION

In this work, we introduced a new control knob for power consumption of DNNs on GPU accelerators. The batch size can control the resource utilization of GPU, and consequently, its power consumption. Based on this control knob we designed the *BatchSizer* approach that can find the proper batch size to maximize the throughput while meeting the power cap, with negligible overhead in a dynamic fashion. *BatchSizer* is orthogonal to previous power management and performance improvement techniques for DNNs such as DVFS, quantization, and pruning, and it can be used in combination with them to yield more promising results. In future works, we aim to explore such combinations.

ACKNOWLEDGMENTS

M. Ebrahimi is partially supported by grants STINT (MG2018-8007) and VR (2016-05140). S. Reda is partially supported by NSF grant 1814920 and DoD ARO grant W911NF-19-1-0484.

REFERENCES

[1] NVIDIA Corporation. 2016. *NVIDIA System Management Interface program.* Retrieved July 27, 2020 from https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf
[2] G Griffin, A Holub, and P Perona. 2007. The caltech-256: Caltech technical report. *vol* 7694 (2007), 3.
[3] João Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomás. 2019. DVFS-aware application classification to improve GPGPUs energy efficiency. *Parallel Comput.* 83 (2019), 93–117.
[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, 630–645.
[5] Andrew Howard, Andrey Zhmoginov, Liang-Chieh Chen, Mark Sandler, and Menglong Zhu. 2018. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. (2018).
[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[7] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).
[8] Qing Jiao, Mian Lu, Huynh Phung Huynh, and Tulika Mitra. 2015. Improving GPGPU energy-efficiency through concurrent kernel execution and DVFS. In *CGO'15*. IEEE, 1–11.
[9] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. 2013. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *2013 IEEE 31st International Conference on computer design (ICCD)*. IEEE, 349–356.
[10] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.
[11] Seyed Morteza Nabavinejad, Hassan Hafez-Kolahi, and Sherief Reda. 2019. Coordinated DVFS and Precision Control for Deep Neural Networks. *IEEE Computer Architecture Letters* 18, 2 (2019), 136–140.
[12] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.
[13] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. 2013. Deep convolutional neural networks for LVCSR. In *IEEE international conference on acoustics, speech and signal processing*. 8614–8618.
[14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural info. processing sys.* 3104–3112.
[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.
[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *IEEE conference on computer vision and pattern recognition (CVPR)*. 2818–2826.
[17] Zhenheng Tang et al. 2019. The impact of GPU DVFS on the energy and performance of deep learning: An empirical study. In *e-Energy '19*. 315–325.
[18] Hadi Zamani et al. 2019. GreenMM: energy efficient GPU matrix multiplication through undervolting. In *ICS*. 308–318.
[19] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*. 8697–8710.