Power-Aware Characteristics of Matrix Operations on Multicores

Guruprasad Konnurmath & Satyadhyan Chickerur

APPLIED ARTIFICIAL INTELLIGENCE
2021, VOL. 35, NO. 15, 2102–2123
https://doi.org/10.1080/08839514.2021.1999013

Taylor & Francis
Taylor & Francis Group

RESEARCH ARTICLE

Power-Aware Characteristics of Matrix Operations on Multicores

Guruprasad Konnurmath[a] and Satyadhyan Chickerur[b]

[a]School of Computer Science and Engineering, K L E Technological University, Hubballi, Karnataka, India; [b]Centre for High Performance Computing, K L E Technological University, Hubballi, Karnataka, India

ABSTRACT
GPU accelerators are massively parallel in nature and tailored for processing numerically intensive high-performance computing applications. But most of the applications that involve heavy computations take excess time to process as the dataset gets larger and lead to more power consumption. Hence, among all the factors in sustainable computing that contribute to operartional costs of GPU, power and time management is one of the major challenging issue. This paper presents a methodology to reduce power consumption in GPUs, meanwhile keeping parallel execution of jobs as high as possible. To achieve this, a power and time aware framework is created by integrating TensorFlow library, InterPSS, and Dynamic Voltage Frequency Scaling (DVFS) techniques. For most of the scientific computing, matrix operations are considered as the fundamental building block. So, the performance, power consumption, and power efficiency of GPU for matrix applications are analyzed under proposed model. Experimental results reveal that proposed methodology substantially reduces peak power of GPUs by 20%, with improved speedup in execution time around 15%.

ARTICLE HISTORY
Received 28 January 2021
Revised 19 October 2021
Accepted 22 October 2021

Introduction

Unfeasible increase in consumption of energy has forced manufacturers of hardware components to prefer energy efficiency as primary concern in their design implementations. The main objective in managing power is to improve performance metrics within the estimated power budget. In case of multicores systems enabled with High-Performance Computing (HPC) capabilities, Graphics Processing Units (GPUs) have proven its performance at peak level as compared with CPUs. In the recent ranking for Top 500 supercomputers, 101 of them are GPU-accelerated systems. But due to its much complex structure with its own memory, control chipset integrated with many processors, high computational power, and capacity to perform much sophisticated task, GPUs consume much higher energy. Some GPUs alone consume 100 watts of power and sometimes more than all other combined parts of the

CONTACT Guruprasad Konnurmath guruprasad.konnurmath@kletech.ac.in School of Computer Science and Engineering, Kle Technological University, Hubballi, India

computing system. To address these issues, especially for the researchers, understanding the mechanisms to cater power management is significantly an important factor to produce effective solutions and introduce power optimized GPUs. Applications related to High-Performance Computing (HPC) mainly utilize GPUs and consume high power as compared to CPUs with much impact on architectural design, reliability, as well as economic feasibility. Many power management strategies such as power aware algorithms (Matteo, Vanzolini, and Mucci 2015) (Steven and Daescu), programming models (Hesham, Moussa., and Farag 2017), CPU-GPU workload division, dynamic resource allocation techniques, and energy saving mechanisms in GPU components (Khaled, El-Hosseini, and Ali 2015) (Sparsh and Jeffrey 2014) have been proposed to tackle power management. Research (Hayri et al. 2016) suggests that more software modifications are required to exploit the resulting improvements in current and future hardware technologies. Basically, TensorFlow is open-source popular programming framework particularly used to perform numerical computations employing dataflow graphs. This TensorFlow also supports several HPC applications (Chien et al. 2019) that run on variety of device types, including GPU and CPU. In this research work, TensorFlow framework is used to notify GPU information such as the GPU type, total number of GPUs, the type of application running on GPU, and its memory consumption. Since TensorFlow library supports pipelining mechanism that creates data parallelism, it is used to reduce execution time.

Along with the analysis of GPU power utilization in terms of various metrics, understanding state of art works for multicore Central Processing Units with different embedded platforms used for similar kind of DSP and AI workloads under specific applications, delivers broader overview about various models, processor architecture, coding strategies with respect to the analysis of power utilization, and performance. For instance, minimal selection of architecture of the deep neural network and software framework of deep learning along with specific embedded platform of hardware show some differences with respect to performance. Mainly few of software integrated deep learning frameworks showing the same functionality do not produce the performance in a similar way when the same model of network is executed on a specific hardware platform. Implementation in the work (Velasco-Montero et al. 2019) illustrate different programming techniques and underlying libraries of acceleration exhibit huge impact on consumption of power, instantaneous throughput, and workload utilization on CPU when the same inference is carried with OpenCV, TensorFlow, Caffe libraries, and Caffe2 techniques on a multicore processor which is ARM Cortex-A53. Also the proposed framework exemplify statistical analysis about how the hardware-oriented resources are differently exploited and also illustrates stating a strong bond in between inference throughput and results, including consumed power and correlated sensitive parameters related with usage of memory modules and procedures flow control. The advent of trending artificial intelligence and machine learning techniques in the field of terrestrial applications have revolutionized new productive innovations in our everyday life the future space mission concepts require high-performance on-board data processing with limited power consumption and enhanced dependability. In this context, the work (Leon et al. 2021) illustrates heterogeneous system of multicore on chip processor for the use on-board future generation spacecrafts to support realistic digital signal processors having computational efficiency and Artificial Intelligence functionalities. To exhibit need and efficiency of lower energy consumption in satellites, implementations include integration of Intel's system on the chip Movidius Myriad2 and mainly focus on software development as well as performance parameters. A systematic methodology and software framework is presented to reach effective partitioning, concurrency, mapping, parallelism, optimization of code, and soft tuning of complex productive algorithms. Along with this, avionics architecture is introduced combining this with field programmable gate array device. The results illustrate several benefits of using Myriad2 in replacement with conventional field programmable gate array and Central Processing Units. In case of Convolutional Neural Networks (CNNs), workloads produce streaming kind of character which make these CNN appealable for reconfigurable mode of architectures that include Field-Programmable Gate Arrays (FPGAs), while there is major need for lower consumption of power and execution speed has introduced Application-Specific Integrated Circuit (ASIC) kind of accelerators as replacement giving alternative effective solutions. The huge improvement in Hardware Description Language (HDL) integrated CNN accelerators, either for Field-Programmable Gate Arrays or Application-Specific Integrated Circuit, has created great academic interest because of high performance and flexible platforms for optimizations. In the view with this, a framework comprising library-based software is proposed in the work (Leon et al. 2020), which incorporates TensorFlow methods, a framework based on machine learning and automatically produces higher effective throughput inference engines of CNN for ASICs and FPGAs. The proposed framework in the work allows developers of software to exhibit the productive outcomes of ASIC/ FPGA acceleration without the need of any expertise on HDL implementation and lower level design. In case of results with comparison of various types of CPU and GPU combinations, the accelerator results around 10 times of improved speedup on the execution of inference engine and optimizable savings in power.

To clearly analyze power features of processing elements running on GPUs, it is better to simulate the undergoing system. Hence, in the proposed methodology, the computing system is simulated using InterPSS (Mike and Huang 2017), a power system simulation technique. The relationship between GPU power consumption and inherent computational patterns is interpreted using dynamic voltage and frequency scheduling (DVFS) technique (Ashish and Khare 2015). To address this issue of power management, we consider two main design features of GPUs that contribute to achieve higher performance: (i) on-core chip massive parallelism based on rate of total number of computation instructions and (ii) memory bandwidth with respect to total count of global memory transactions. GPU kernel's performance is primarily dependent on the frequency of these two operating components. Two applications, namely, computationally intensive matrix multiplication and memory intensive matrix transpose are used to characterize energy efficiency of GPUs. The common data access patterns and heavy parallel computational requirements suggest matrix operations as best suitable for effective evaluation on GPUs.

The combination of TensorFlow, InterPSS, and DVFS techniques in the proposed methodology make GPU to perform computations in a very efficient and optimized way with speedy execution and lesser power consumption. To achieve this following objectives are defined:

- Analyze and investigate power dissipation in association with parallel program execution of multi-core processor/GPU in a computer system.
- For analysis, identify possible areas of various scientific applications, techniques, and automate the process of workload power analysis.
- Design & implement time aware model that exhibits speedy execution and power aware model to optimize power consumption.
- Model relationship between time and power characteristics and integrate to result workload aware model to achieve both time and power efficiency in multi-core processor/GPU. Finally compare and evaluate performance.

Background

In the past decade, many power management techniques, algorithms, and programming models have been introduced by various researchers. To have clear review about the research work, literature survey with respect to time and power management in multicores and GPUs is carried out under various categories.

For power management, the techniques are broadly classified among two major categories: reactive and predictive (Khaled, El-Hosseini, and Ali 2015). The reactive techniques react in accordance to changes in performance based on workload. When the change is identified in workload's state, accordingly this technique responds to that specific change. Hence, predictive technique is preferred to be better than reactive technique. Our research work is followed by predictive technique. To optimize power consumption in CPU as well GPU, the most well-known technique is DVFS method (Sparsh and Jeffrey

2014)(Ashish and Khare 2015)(Xinxin et al. 2013). This method is implemented by altering the voltage and frequency levels of multicores. This technique is introduced commercially under many names such as AMD's PowerNow, Intel's SpeedStep technology and many more.

Improper workload distribution between CPU and GPU too affects the energy consumption. Several research works have been carried out to compare and validate the energy consumption between CPU and GPU. The survey results (Sparsh and Jeffrey 2014) show GPUs consumes more energy as compared to CPU. The methodologies introduced under some of the frameworks (Dong, Byna, and Chakradhar 2011) prove that managing GPU workload consolidation systematically can improve overall throughput of the system leading to desirable savings in the energy as compared with multicore CPUs.

Power consumption on any GPU varies depending on the type of application introduced. Hence, selection of proper applications (Phuong, Young., and Gun 2017) and its implementation is one of major factor to analyze energy efficiency in GPUs. Rodinia benchmark suite (Martin, Giusti, and Naiouf 2018) an application set that includes PathFinder, SRAD, BFS, LavaMD, CFD Solver, and LUD were mainly used to interpret the performance, energy consumed by CPU along with GPUs. Mainly the suite compares single-threaded and multi-threaded versions of CPU with GPU implementations and characterizes instant power generated, total time executed, and average energy consumed. Some of the experiments were implemented on real-time GPU platform tested using 37 benchmark applications (Xinxin, Wang, and Chu 2016). Results revealed that by gradually scaling down the GPU core voltage and GPU core frequency, around 19.28% energy reduction can be achieved and prove GPU DVFS (Ashish and Khare 2015) to be one of the effective approach for energy conservation in GPUs. Power profiling also varies for different types of GPU architectures. By altering core clock as well as memory clock frequencies experiments (Peter et al. 2016,) were conducted for three Nvidia GPU generations (Maxwell, Fermi, and Kepler architectures). The outcomes prove that the change in architecture of GPU also has an impact on power consumption.

Selection of proper algorithms and programming languages helps in efficient utilization of resources by the GPUs. A detailed study of memory utilization, energy consumption, and runtime of well-known software languages is discussed in the paper (Rui, Couto, and Cunha et al. 2017). Results reveal slower programming languages consume more energy and a faster language consume less and later analyzes the influence of memory usage on energy consumption. Also some compilers supporting these programming languages not only influence code's execution performance but also affect the energy consumption. Even though parallel algorithms are effective some of them fall short in terms of optimization when addressing the issue of unique

GPU architectures. The design model (Steven and Daescu) for a parallel algorithm using template matching algorithm, introduces Parallel GPU Model (PGM), show better degree of optimality for GPU architecture compared with traditional type parallel models. The increasing demand of GPU usage in HPC systems for computations has led to major challenges in power management. To address these power management issues, machine learning methods provide software-based unique approach. The problem of GPU power management is explored using machine learning at different DVFS states. An ensemble technique (Bishwajit, Adhinarayanan, and Feng 2018) integrated with three techniques of machine-learning namely sequential minimum optimization regression, linear regression, and decision tree were implemented to decrease mean absolute error around 3.4%. These three techniques also examine energy management in GPUs. Several examples (Martin and Barham et al. 2016) were introduced and analyzed to describe the efficiency of TensorFlow programming model that facilitates unique experimentation and proves that the resulting implementations are scalable and performant. The results interpret the advantages of running well-known operations in deep learning such as matrix calculations and convince TensorFlow libraries are flexible for both CPU and GPU architectures parallelly.

Basically simulations are performed to observe and analyze dynamic behavioral nature of the objects and predicts about the change in functioning of the entire system when the individual components of that system are being altered. The detailed survey of power modeling in GPUs and profiling methods (Robert, Imam., and Mintz 2016) like Qsilver, UNISIM, PowerRedMcPAT implemented on GPUs show that software-based power measurement tools, computer power monitoring have grown at a larger extent, which is made possible by modeling and simulation environment. A power model for GPU using McPAT and a CPU power simulation tool is developed that estimates the power consumed by different components of GPU by using configuration parameters, which can be determined through experimental evaluation. The design of InterPSS simulation engine (Mike and Huang 2017), including its software architecture, object model, and development process of software prove InterPSS to be more flexible and efficient simulation software. To perform power analysis of a system, components from existing system and other system can be integrated into InterPSS. In the proposed research work, InterPSS is used to provide power system analysis/simulation model service.

Experimental Methodology

The proposed methodology's architecture is shown in Figure 1. This methodology is carried at two levels. First at the hardware level, system component power analysis is performed using TensorFlow, InterPSS and DVFS. In the second, software function analysis is performed using TensorFlow library

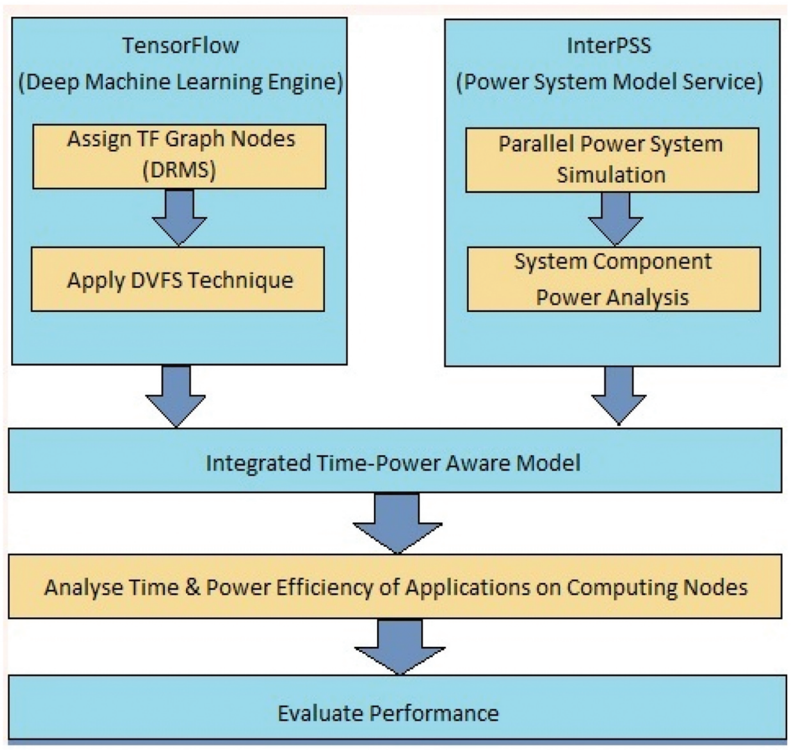
TensorFlow
(Deep Machine Learning Engine)
Assign TF Graph Nodes (DRMS)
Apply DVFS Technique
InterPSS
(Power System Model Service)
Parallel Power System Simulation
System Component Power Analysis
Integrated Time-Power Aware Model
Analyse Time & Power Efficiency of Applications on Computing Nodes
Evaluate Performance

Figure 1. Architecture of proposed methodology.

to achieve parallel computation involving better concurrency. Both levels are integrated to form time-power model. Efficiency of execution time and power consumption is analyzed using matrix multiplication and matrix transpose applications. Finally, performance is evaluated by comparing the GPU implemented with the proposed methodology to the naive GPU without any such methodology.

TensorFlow and InterPSS

TensorFlow engine performs activity of dynamic resource management system (DRMS) by generating graph nodes. Nodes of the graph are represented by the mathematical operations and the edges in the graph communicating between these nodes represent multidimensional arrays called as tensors. This flexible and adaptive architecture of TensorFlow allows distributing computations on one or more CPU as well GPU in par with the requirement. Representing itself as a graph, an efficient data structure TensorFlow allows boosting execution speed. Whenever, the unused nodes in the graph are detected and eliminated, thus optimizing it for size and evacuating idle power consumption. It also identifies redundant operations or sub-optimal graphs and replaces them with the best alternatives with the aim of optimizing time constraints. This nature of TensorFlow ensures computation optimization yielding efficiency in terms of execution time as well as power consumption. In the proposed work, TensorFlow library is used to run matrix operation on particular device instead of automatic selection using *tf.device* to create a dedicated device context and all the related operations around the context shall execute on the same designated device. Since integrated GPU does not own video Random Access Memory, this integrated GPU requires only a minimal amount of memory space. In comparison with onboard-graphics card consist of its own video memory module, or a short Video Random Access Memory, which is found to be one of the biggest advantage. Also for these dedicated graphics card, peripheral, and external devices are clocked in faster way leading to high-performance level. This huge performance throughput is accompanied with higher power consumption, heat dissipation and which lead to memory fragmentation in some cases. TensorFlow library in the proposed work is used to select the dedicated devices and to limit memory growth.

TensorFlow whenever programmed for GPU by default it maps to nearly all the GPU memory of all available GPUs (with respect to *cuda_visible_devices*) This process is undergone to efficiently utilize the precious GPU memory resources relatively on the devices for reducing the increasing ratio of memory fragmentation. To limit the TensorFlow utilization for a specific set of GPUs among all available GPUs, the method *tf.config.set_visible_devices* is used. In the implementation, memory growth is turned on by using *tf.config.experimental.set_memory_growth* that allocates only memory as required by GPU based on its computation. As the method is called it begins to allocate very little memory, and when the program starts executing more amount of GPU memory is allocated and extended for this TensorFlow process. Because it may lead to memory fragmentation, memory is not released continuously at a stretch. Based on above mentioned conditions and advantages with respect to dedicated GPU, in the proposed work, we apply Tensorflow method only for prioritizing GPU device placement and limiting GPU memory growth and not for NN framework for building and running the model. TensorFlow libraries *tf.data, num_parallel_calls* and *tf.data.experimental.AUTOTUNE* allows to increase data parallelism and automatically improves the execution speed.

Power measurement, analysis of jobs, and system components is done using Internet technology based Power System Simulator(InterPSS). InterPSS architecture enables components associated with GPU to be easily plugged into InterPSS to augment its functionality and perform any kind of power analysis. Hence, for power analysis, InterPSS simulates existing system and provides service to the TensorFlow deep learning engine. Since it's able to map the memory of almost all the GPUs visible to the processes, TensorFlow relatively uses the GPU memory resources efficiently there by reducing the memory fragmentation on the devices. In the proposed work, this simulation being applied to the existing system integrated with GPU GeForce GTX 1060 shall be helpful in enhancing power system design, clear analysis, diagnosis, and operation of power flow without any power flow variation, when the DVFS technique is applied to alter voltage and frequency levels as compared to the scenario without application of InterPSS. Also this helps to identify the memory usage of each GPU including details of its idleness and workload share.

DVFS

The main priority to use DVFS technique in this paper is to analyze and optimize power consumption of GPU by reducing voltage or frequency during the interval when the GPU has lesser workload. Due to limitation of applications parameters, it is not always reasonable for any kind of application requiring GPUs to map and utilize all the available cores. In many of the applications, sometimes the memory bandwidth of GPU cores acts as a bottleneck affecting the throughput and performance of GPU. Because of this affecting bottleneck, the GPU cores remains unutilized many number of times during its running process, an efficient power management technique is needed. Based on the status of GPU workload gathered using TensorFlow, that informs whether the application on GPU has lesser-activities undergoing or idle periods, power consumption may be reduced by applying DVFS technique on those GPUs. Based on the status of GPU workload gathered using TensorFlow, that informs whether the application on GPU has lesser-activities undergoing or idle periods, power consumption may be reduced by applying DVFS technique on those GPUs. In this research work DVFS is implemented by altering the frequency levels of GPU core and GPU memory. Following equation depicts the energy consumed by GPU:

$$Power = Capacitance * Voltage^2 * Frequency$$

where,

- P = Power consumed by GPU
- C = Capacitance
- V = Voltage supply to GPU
- F = GPU clock frequency

Hence, the power consumed by any application applied with DVFS can be decreased by reducing Frequency or Voltage, or both. The main preference to implement DVFS technique in the paper is to analyze and optimize GPU power consumption by reducing voltage and frequency during the periods when GPU undergoes lesser workload. As the frequency is lowered, the consumption in power will be lowered parallelly. And if there is lowering of voltage is done, there is a drop in consumption of power. A strong correlation exists between the frequency in clock and performance, which indicates that decrease in frequency also leads decrement in the performance. Hence as a first priority to reach these issues, DVFS mainly provides tool to improve performance with power and energy reduction. DVFS technique is largely used to achieve Energy per instruction at different ratios. The main goal of this DVFS technique is to adjust frequency and voltage pairs within separate yet already defined pairs to achieve optimized power consumption and improved level of performance. For huge parallel workloads, multiple cores execute at lower rate of voltage and frequency pair. In case of scalar workloads that includes larger portion of serial code, it is proved better to run on few cores and improve their frequency to adjust to the specific task. To implement DVFS technique in the proposed work, mainly two predefined performance levels is fixed for the GPU (NVIDIA GeForce GTX 1060), namely 'Idle' and 'Maximum Performance.' In case of the maximum performance settings, clock speeds of the GPU is set to highest level to reach the best effective performance. For idle setting level, when the TensorFlow method detects the GPU is in the idling state, then automatically it will low down the GPU cores frequencies and memory to the already defined 'Idle' level. Specifically, the NVIDIA GeForce GTX 1060 GPU core clock is set from 600 MHz to 800 MHz mean, while the GPU memory clock is set from 600 MHz to 1000 MHz.

Experimental Setup

The aim of the proposed research work presented is to examine and analyze GPU kernel's power consumption, execution time, and performance for matrix operations for particular intervals. Data is collected frequently from multiple runs of matrix operations and since the focus of research work is on power analysis of GPU alone, only the data related to that GPU is included. The research work is implemented on a NVIDIA's GeForce GTX 1060 GPU card integrated with 6 GB GDDR5 memory type hosted by Intel. It has supporting graphics clock frequency of 1500 MHz and processor clock frequency around 1700 MHz with 1280 cuda cores. The minimum system power requirement is 400 watts and maximum power consumption of GPU is around 120 watts. The software configuration includes ubuntu distribution installed with CUDA toolkit 10.0. Hazelcast Python Client library is used to integrate TensorFlow Python runtime environment and InterPSS Java runtime environment for communication.

Application Characterization

Matrix multiplication and matrix transpose applications are implemented under the integrated framework of TensorFlow, InterPSS, and DVFS techniques to analyze time and power efficiency of GPUs in the configured system. In our experiments, two matrix applications were tested, first one represent dense matrix multiplication considered to be computationally intensive application and the second dense matrix transpose found to be memory intensive application to showcase consumed power and performance characteristics of the GPU. Following section gives detailed description of the two tested applications.

Dense Matrix Multiplication

This dense matrix multiplication (MatMul) application is systematically optimized in a way to use the maximum computational power characteristics of GPU. To take the advantage of coalesced global memory accessing of GPU and faster local memory, blocked version of matrix multiplication algorithm is adopted. The rate of instructions issued (Mike and Huang 2017) by the GPU kernel is the major bottleneck to be handled. It offers memory access regularly and heavy computations parallelly but features data reuse of O(n) and conforms to be the best legitimate candidate for faster implementation of GPUs.

Dense Matrix Transpose

Matrix Transpose is a building block algorithm for many applications that performs conversion of array of rows M by columns N (i.e. M*N) to array rows N by columns M (i.e. N*M). Whenever the offload of algebraic libraries to GPUs is high, increased performance for in-place transposition is required. Hence, this in-place transpose have to be best fit for most of GPU architectures because of its minimal availability of on-board memory and maximum throughput. The dense matrix transpose (MatTran) preferably designed for memory related manipulations with lesser amount of required computation for memory indices and thread's ID. To completely utilize GPU's capability of parallel processing, every multiple rows of matrix are simultaneously interpreted. Intermediate outcomes are recorded in local memory, holding them to write back into global memory.

Experimental Profiling

To understand the execution of the kernel, profiling information of each kernel is collected through NVIDIA's OpenCL Visual Profiler. To understand whether a kernel is of memory-intensive type or of compute-intensive these two types of kernel is understood on the basis of rate of instruction issues for the first type and second type defined by ratio of number of global memory transactions to computation instructions. A compute intensive task is found to be considered as the task to fully utilize the potential computational power effectively and requires a large amount of computation on GPU cores. In case of memory intensive task, an application requires a large amount of memory for computing a particular application. In the proposed work, as mentioned in the section power aware matrix multiplication and power aware matrix transpose, for matrix multiplication the narrow growth of performance in Figure 2 mentioned under the section Results and Discussions, at all frequency levels show that the performance of GPU for matrix multiplication is independently determined by core frequency of the GPU. This is mainly due to higher rate of instructions are executed on GPU cores at all frequencies. Hence, matrix multiplication is considered to be performing compute intensive task. In case of matrix transpose, as shown in Figure 3 mentioned under the section Results and Discussions, the memory clock of the GPU determines the performance and efficiency of matrix transposition and found to be independent from its core frequency. This complete memory oriented dependency of matrix transpose is decided based on the rate of global memory transactions. Hence, matrix transpose is considered to be performing memory intensive task.

Table 1 gives details about GPU kernel categories, the type of applications used and average power consumed in watts. Based on the computational ability matrix multiplication is considered compute intensive application and matrix transpose as memory intensive application.

The memory efficiency ($Rat_{Mem)}$ is calculated by total number of memory transactions of instructions taken place in GPU memory upon total number of instructions computed on the GPU core.

$$\begin{matrix} Rat \\ Mem \end{matrix} = \frac{Number\ of\ Global\ Memory\ Transactions}{Number\ of\ Computation\ Instructions}$$

The core efficiency ($Rat_{Instrns}$) of GPU core is calculated by computing total number of instructions computed by GPU core upon time taken by GPU to compute instructions as shown below:

$$\underset{Instrns}{Rat} = \frac{Number\ of\ Computation\ Instructions}{Time\ taken\ by\ GPU}$$

The profiling information of Rat_{Mem}, $Rat_{Instrns}$ for each of the two applications were developed, with the sample codes available from, as mentioned under Table 2. Periodically, the total number of global memory requests and also the transactions has to be reported since multiple global memory transactions might be required for single global memory request. Hence, this is nothing but the actual count of all the memory transactions that finds the latency for each memory access. Each request on memory will be considered as an instruction that is deducted while determining the real number of instructions computed.

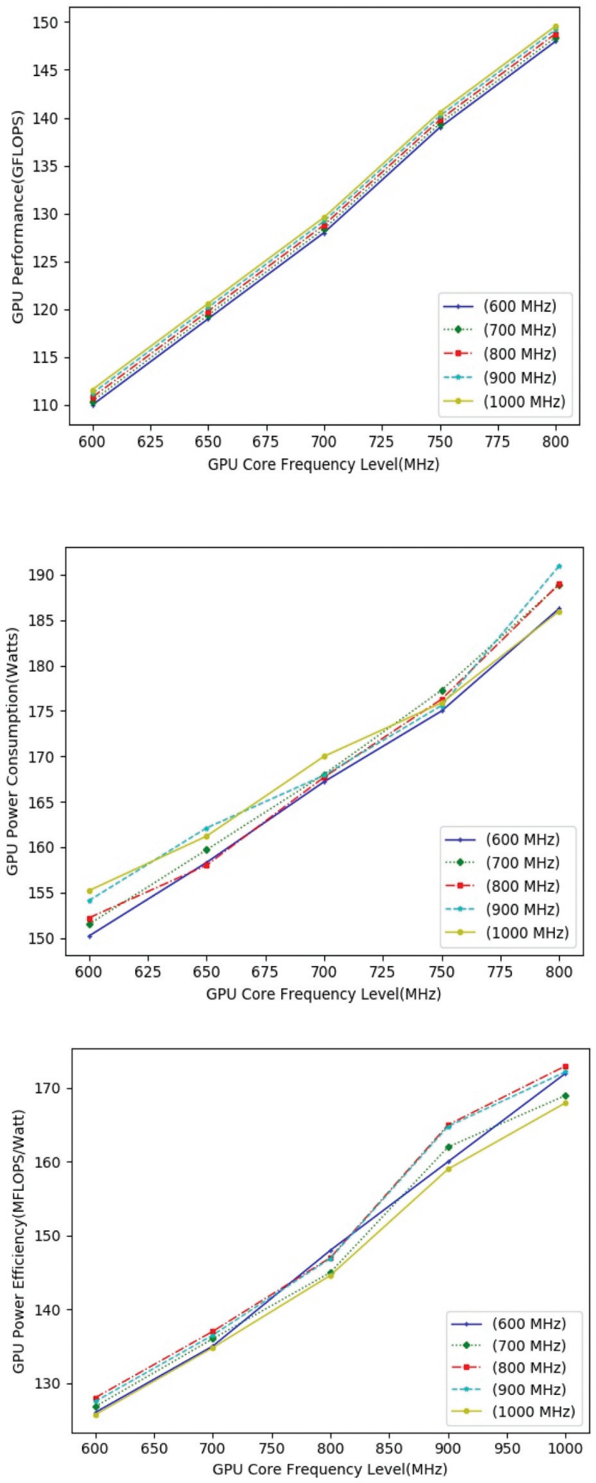

Figure 2. Evaluation of performance, power consumption, and power efficiency for matrix multiplication.

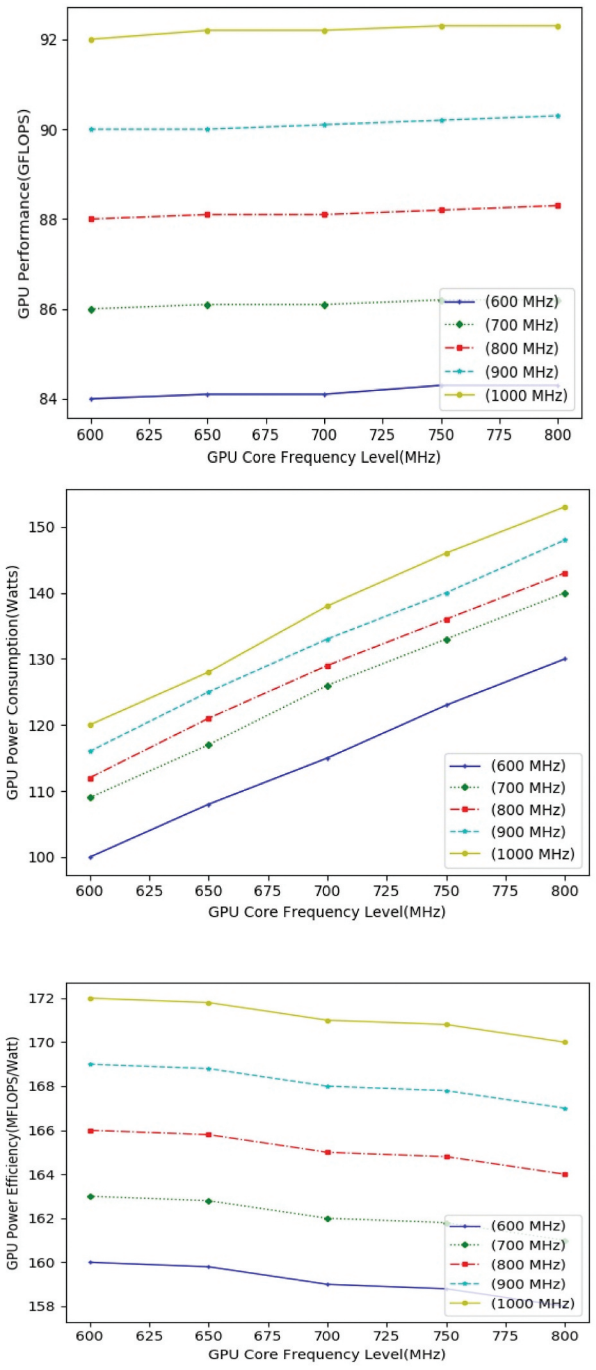

Figure 3. Evaluation of performance, power consumption and power efficiency for MatrixTranspose.

Table 1. Kernel category and application type.

Kernel Type	Application Type	Average Power Consumption
Compute Intensive	Matrix Multiplication	150 w
Memory Intensive	Matrix Transpose	170 w

Table 2. Application characterization.

Application Type	Rat_{Mem}	$Rat_{Instrns}$
Matrix Multiplication	4.52%	203102301
Matrix Transpose	43.7%	10084784

Evaluation Metrics

Following evaluation metrics are used to compare and analyze various DVFS configurations of the system for each application:

- **Time**: The measurement of each application's kernel execution is the execution time. To reduce the effects of noise, application of same type is run for multiple number of times for each setting and also average minimal time (T) is used.
- **Performance**: Most well known quality assurance metric used in HPC systems is Floating-Point Operations per Second (FLOPS), but this performance metric cannot be applied for some specific applications such as matrix transposition. Due to another metric megabytes per second (MBPS) is used to analyze the performance of matrix transpose to notify about the throughput during the run time.
- **Energy**: Energy (E) is major metric measured regularly during particular intervals for the entire system while executing GPU kernel.
- **Power**: Usually considered as average power (P), is calculated by ratio of total amount of energy consumed upon time taken for execution:

$$Power = \frac{Energy\ Consumed}{Execution\ Time}$$

- **Power Efficiency**: Power efficiency is indicated by the ratio of improved performance on each GPU per power.

$$Efficiency\ ofPower = \frac{Improved\ Performance}{Power}$$

Results and Discussion

Under this final section, experimental results for the applications matrix multiplication and matrix transpose is presented fewer than two categories. The first category defined by power aware model presents the results obtained by measuring the metrics to analyze power consumption, power efficiency, and performance for each application running on the GPU. And the second category defined by time aware model presents the improved execution speedup of GPU kernel under proposed methodology for both applications in comparison with naive GPU method.

Power Aware Matrix Multiplication

The power usage of the system without GPU utilization in the idling state is 111.2 watts and 142.5 watts for the GPU enabled system.

Separate set of graphs are generated for the proposed methodology to present performance, power consumption and power efficiency of dense matrix multiplication, as shown in Figure 2. The x-axis in graph depicts core frequency of GPU and the legend plots memory frequency of GPU. The matrix multiplication application is run under different GPU core frequencies (600 MHz, 650 MHz, 700 MHz, 750 MHz, and 800 MHz) and different memory frequencies (600 MHz, 700 MHz, 800 MHz, 900 MHz, and 1000 MHz) to reach reduction in power consumption. The narrow growth of performance on Figure 2, at all frequency levels show that the performance of GPU for matrix multiplication is independently determined by core frequency of the GPU. This is mainly due to higher rate of instructions are executed on GPU cores at all frequencies with much greater speed using TensorFlow library and much lower rate of global memory transactions at GPU memory.

As depicted in the Table 2, around 2 million instructions per computing unit are executed and the rate of global memory transactions is not more than 5%. Hence, the matrix multiplication for the proposed methodology executed under GPU proves to be compute intensive application and to achieve better performance.

GPU has to be configured with highest core frequency and lowest memory frequency. Also, the high level data parallelism of TensorFlow library boosts the execution speed by double for different frequencies. Due to this, around 20% reduction in power consumption is achieved with improvement in power efficiency by approximately 18 MFLOPS/watt.

Power Aware Matrix Transpose

Behavior of matrix transpose application kernel is shown in Figure 3 for different GPU core as well as frequency settings. The group of horizontal lines seen in the graph show that the memory clock of the GPU determines the performance and efficiency of matrix transposition and found to be independent from its core frequency. This complete memory oriented dependency of matrix transpose is decided based on the rate of global memory transactions obtained by the ratio of Rat_{Mem} using TensorFlow library having much lesser effect on GPU core frequency. The global memory transaction intensity of GPU kernel for matrix transpose is 20 times greater than matrix multiplication intensity. Hence to reach better performance in terms of power optimization for matrix transpose, GPU should run at maximum memory frequency and minimum core frequency.

We can analyze from Figure 3 that GPU is run and processed at GPU memory frequency with highest level as well as GPU core frequency with lowest level to reach nearly optimized performance mean while with reduction in the absolute power been consumed by approximately 3 watts for every decrement in frequency of 50 MHz and indicate computing units idle for most of times for matrix transpose kernel. Further in comparison with matrix multiplication, the power consumption is less for matrix transpose with improvement in power efficiency by 15%. When memory frequency of GPU is fixed constant at 1200 MHz, minimal power consumed to execute matrix transposition is around 216.4 watts and maximum power observed around 237.3 watts, means just a few difference of 20 watts.

Time Aware Model

The following section presents comparison of GPU speed between GPU kernel implemented with the proposed methodology and the naive GPU without any such approach. The proposed work in this time aware model mainly project on higher implementation of parallelism for Matrix operations. Basically, these matrix operations are widely used in many areas of scientific computing communities and also the basis for mathematical operations on multicore processors. Proposed solution for Matrix operations is achieved by exploiting the higher level of parallelism by the multicore GPUs. It adopts to several generalized GPU effective workload optimizations and also specific type of GPU architectural, design optimizations. The heterogeneous higher level parallel matrix operations kind of method is tested for matrices under different sizes and various ranges of power. Proposed work uses the highly optimized GPU for the matrix multiplication and matrix transpose. Because the matrix-related operations possess heavy data parallelism nature of applications, matrix operations exhibit maximal speed up in a fine tuned concurrency and massive parallelism supported devices like Graphics Processing Units.

Both methods are compared with various sized matrices and powers in the increasing for matrix multiplication and matrix transpose applications. Table 3 shows speedup comparison between naive GPU kernel and proposed methodology for matrix of 64 by 64 size. The naive GPU kernel has good performance speedup of almost 4 times but the speedup remained constant, even though the power of matrix increased exponentially. In case of proposed methodology for matrix multiplication and matrix transpose show high-performance improvement and increased speedup correspondingly with the increase in matrix power exponentially over the naive GPU approach. The graphical representation of speedup comparison between naive GPU kernel and proposed methodology for the matrix of size 64 by 64 is shown in Figure 4.

Table 4 shows the comparison of speedups in seconds between naive GPU kernel and proposed methodology for matrix size 128 by 128. As shown in the Figure 5, the speedup comparison represented using graphs depicts that the naive GPU has better speedup in performance of almost 6 times for matrix size of 128 by 128. After some time, the speedup remained constant even though

Table 3. Matrix exponentiation of size 64 by 64.

Size	Naive GPU(MatMul) inseconds	Naive GPU(MatTran) inseconds	Proposed Methodology(MatMul) inseconds	Proposed Methodology(MatTrans) inseconds
64	0.1	0.15	0.04	0.06
128	0.3	0.35	0.04	0.06
256	0.7	0.8	0.05	0.075
512	1.2	1.5	0.05	0.075
1024	2	2.4	0.06	0.082

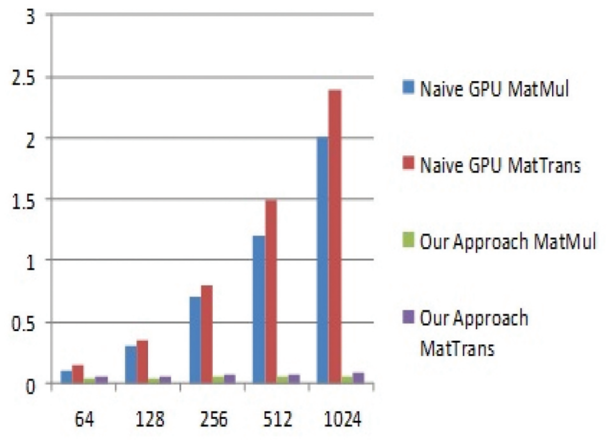

Figure 4. Speedup comparison for matrix size 64 by 64.

Table 4. Matrix exponentiation of size 128 by 128.

Size	Naive GPU(MatMul) inseconds	Naive GPU(MatTran) inseconds	Proposed Methodology(MatMul) inseconds	Proposed Methodology(MatTrans) inseconds
64	0.1	0.11	0.02	0.03
128	0.3	0.33	0.021	0.04
256	0.4	0.5	0.023	0.055
512	0.7	0.85	0.023	0.06

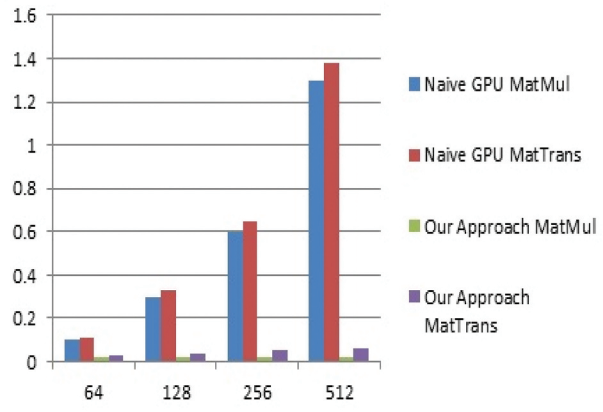

Figure 5. Speedup comparison for matrix size 128 by 128.

there is an exponential increase in power of the matrix. Hence, the proposed approach for matrix multiplication and transposition not only display the larger improvement in performance over naive GPU approach, even it shows increased speedup correspondingly with the increase in matrix power exponentially.

Based on comparison obtained from Table 5 and the graphical representation in Figure 6, the power of matrix has increased exponentially but the speedup remained constant for naive GPU approach. Hence, it is clear that, the matrix of 256 by 256 size with maximum exponential value for matrix power 512 exhibit enormous improvement of 10 times speedup for the proposed methodology as compared with the naive GPU kernel code.

As shown in Table 6 and Figure 7, along with greater performance improvement over the naive GPU approach, the proposed methodology for matrix multiplication and matrix transpose shows increased speedup correspondingly with the increase in power of matrix exponentially. The

Table 5. Matrix exponentiation of size 256 by 256.

Size	Naive GPU(MatMul) inseconds	Naive GPU(MatTran) inseconds	Proposed Methodology(MatMul) inseconds	Proposed Methodology(MatTrans) inseconds
64	0.23	0.4	0.1	0.15
128	0.4	0.5	0.11	0.17
256	0.8	1.0	0.12	0.18

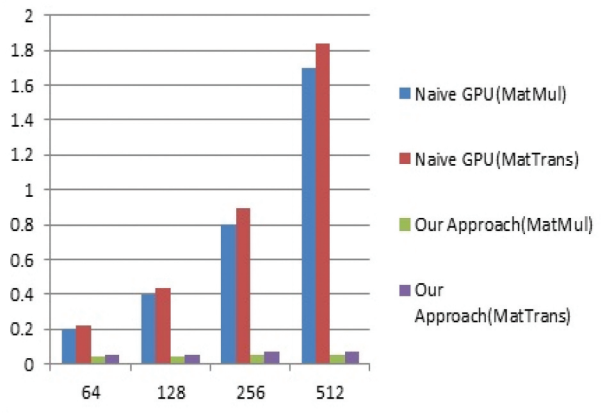

Figure 6. Speedup comparison for matrix size 256 by 256.

Table 6. Matrix exponentiation of size 512 by 512.

Size	Naive GPU(MatMul) inseconds	Naive GPU(MatTran) inseconds	Proposed Methodology(MatMul) inseconds	Proposed Methodology(MatTrans) inseconds
64	0.2	0.22	0.042	0.05
128	0.4	0.44	0.043	0.054
256	0.8	0.89	0.05	0.067
512	1.7	1.84	0.054	0.068

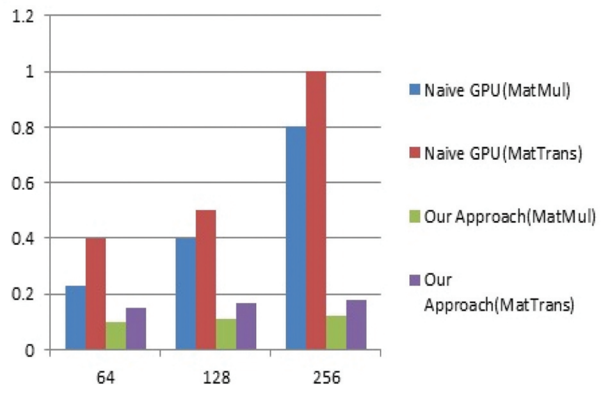

Figure 7. Speedup comparison for matrix size 512 by 512.

matrix of 512 by 512 size with maximum exponential value for the matrix power 256 exhibit greater improvement of 12 times speedup for the proposed methodology as compared with the naive GPU kernel code.

Conclusion and Future Work

The implementation of the proposed methodology on GPUs with the combination of three techniques namely TensorFlow, InterPSS, and DVFS supports the capability of supercomputing with relatively cheaper GPUs and from the results we can analyze that the performance, power efficiency, and power consumption of GPU application kernels are determined by the rate of instruction issues by the GPU cores and the ratio of number of global memory transactions to the total number of computation instructions by GPU memory. The proposed methodology is tested for dense matrices up to 512 by 512 size against the high powers of up to 256 that show reasonable tremendous improvement in execution speed around 15%. TensorFlow's flexible dataflow architecture allows power users to achieve excellent performance and supports automatic GPU placement, GPU kernels fusion, efficient GPU memory management, and scheduling that can be considered as a better alternative machine learning technique for power optimization and speedy execution. Also, the proposed approach includes many architectural performance benefits confined to Nvidia GTX series GPUs. Also, pooling of DVFS technique with TensorFlow engine and InterPSS allows reasonable saving of energy in a more optimized way as compared with other energy saving mechanisms. The combined methodology can be further extended for future research work in designing energy efficient Green GPUs and implemented for different GPU architectures.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

Ashish, M., and N. Khare. 2015. Analysis of DVFS techniques for improving the GPU energy efficiency. *Open Journal of Energy Efficiency* 4 (4):77–86. doi:10.4236/ojee.2015.44009.

Bishwajit, D., V. Adhinarayanan, and W. Feng. 2018. GPU power prediction via ensemble machine learning for DVFS space exploration. *ACMDigital library* ISBN: 978-1-4503-5761-6.

Chien, S. W. D.,Stefano Markidis, Vyacheslav Olshevsky, Yaroslav Bulatov, Erwin Laure, Jeffrey S. Vetter. 2019. An evaluation of tensorflow performance in HPC applications, arXiv:1903.04364v1 [cs.DC] March 11, 2019

Dong, L., S. Byna, and S. Chakradhar. 2011. Energy-Aware workload consolidation on GPU. *IEEE* 1530-2016/11 © 2011. doi:10.1109/ICPPW.2011.25.

Hayri, A., G. Alptekin, J. P Gelas, and P. Ghodous. 2016. The impact of source code in software on power consumption. *International Journal of Electronic Business Management, Electronic Business Management Society* Taiwan, 14, pp.42–52. <hal-01496266>.

Hesham, H. M., A. S. Moussa., and I. Farag. 2017. Performance vs. Power and energy consumption: impact of coding style and compiler. *(IJACSA) International Journal of Advanced Computer Science and Applications* Volume 8 (Number 12).

Khaled, M. A., A. M. El-Hosseini, and A. H. Ali. 2015. Dynamic power management techniques in multi-core architectures: A survey study. *Production and Hosting by Elsevier* 2090–4479, Volume 8, Issue 3. Ain Shams University.

Leon, V., G. Lentaris, E. Petrongonas, D. Soudris, G. Furano, A. Tavoularis, and D. Moloney. 2021. Improving performance-power-programmability in space avionics with edge devices: VBN on Myriad2 SoC. *ACM Transactions on Embedded Computing Systems (TECS)* 20 (3):1–23. doi:10.1145/3440885.

Leon, V., S. Mouselinos, K. Koliogeorgi, S. Xydis, D. Soudris, and K. Pekmestzi. 2020. A TensorFlow extension framework for optimized generation of hardware CNN inference engines. *MDPI Technologies* 8 (1):1–15. doi:10.3390/technologies8010006.

Martin, A., P. Barham, et al., 2016. TensorFlow: A system for large-scale machine learning, *OSDI'16 Proceedings of the 12th USENIX(Advanced Computing Systems Association)conference on Operating Systems Design and Implementation, Savannah, GA, USA*, 265–83. ACM Digital library.

Martin, P. P., L. D. Giusti, and M. Naiouf. 2018, October. Are GPUs non-green computing devices? *Journal of Computer Science & Technology* Volume 18 , Number 2.

Matteo, C., L. Vanzolini, and C. Mucci. 2015, March. Power-aware job scheduling on heterogeneous multicore architectures. *IEEE Transactions on Parallel and Distributed Systems* 26 Issue 3, Page(s): 868 - 877.

Mike, Z., and Q. Huang. 2017. InterPSS: A new generation power system simulation engine. 2017 *Link: https, ResearchGate.*

Nvidia CUDA C Programming Guide, version 3.1. 2007. NVIDIA corporation. Link: http://developer.nvidia.com/object/cuda.html .

Nvidia CUDA Programming Guide, Nvidia, Santa Clara, CA, USA. 2011.

Peter Goldsborough , 2016, A Tour of TensorFlow, *arXiv*: 1610.01178v1 [cs.LG], October 1st. https://www.tensorflow.org/guide/gpuhttps://www.nvidia.org

Phuong, T. Y., L. D. Young., and L. J. Gun. 2017. Impacts of optimization strategies on performance, power/energy consumption of a GPU based parallel reduction. *Journal of Central South University Springer* 24:2624–37.

Robert, B., N. Imam., and T. Mintz. 2016, December. Understanding GPU power: A survey of profiling, modeling, and simulation methods. *Journal, ACM Computing Surveys (CSUR)* 49 (Issue 3). Article No. 41,pp 1–27.

Rui, P., M. Couto, J. Marco Couto, Rui Pereira, Francisco Ribeiro, Rui Rua, Jacome Cunha, Joao Paulo Fernandes, . 2017. Energy efficiency across programming languages. *Association for Computing Machinery, ACM ISBN* 978-1-4503-5525-4/17/10. doi:10.1145/3136014.3136031.

Sparsh, M., and S. Jeffrey. 2014, July. A survey of methods for analyzing and improving GPU energy efficiency. *ACM Computing Surveys* Volume 47, Issue 2 . Article 19, Pages 1-23.

TOP500 Supercomputer Site. 2017. http://www.top500.org

Tyler, A., and G. Rong. 2016. Characterizing power and performance of GPU memory access. *IEEE* 978-1-5090-3856-5/16, 2016. Pages 46–53.

Velasco-Montero, D., J. Femández-Bemi, R. Carmona-Gálán, and A. Rodríguez-Vázquez. 2019. On the correlation of CNN performance and hardware metrics for visual inference on a low-cost CPU-based platform. *International Conference on Systems, Signals and Image Processing (IWSSIP)* 249–54. doi:10.1109/IWSSIP.2019.8787329.

Xinxin, M., L. S. Yung, K. Zhao, and X. Chu. 2013. A measurement study of GPU DVFS on energy conservation. *ACMDigital Library* ISBN: 978-1-4503-2458-8.

Xinxin, M., Q. Wang, and X. Chu, 2016. A survey and measurement study of GPU DVFS on energy conservation, *arXiv*: 1610.01784v1 [cs.DC] 6.