The Journal of Supercomputing (2022) 78:14919–14939
https://doi.org/10.1007/s11227-022-04473-9

Investigating the effect of varying block size on power and energy consumption of GPU kernels

Muhammad Jawad Ikram[1] · **Mostafa Elsayed Saleh**[2] · **Muhammad Abdulhamid Al-Hashimi**[2] · **Osama Ahmed Abulnaja**[2]

Accepted: 21 March 2022 / Published online: 10 April 2022

Abstract

Power consumption is likely to remain a significant concern for *exascale* performance in the foreseeable future. In addition, *graphics processing units (GPUs)* have become an accepted architectural feature for exascale computing due to their scalable performance and power efficiency. In a recent study, we found that we can achieve a reasonable amount of *power* and *energy* savings based on the selection of algorithms. In this research, we suggest that we can save more power and energy by varying the *block size* in the *kernel configuration*. We show that we may attain more savings by selecting the optimum *block size* while executing the workload. We investigated two kernels on NVIDIA Tesla K40 GPU, a Bitonic Mergesort and Vector Addition kernels, to study the effect of varying block sizes on GPU *power* and *energy* consumption. The study should offer insights for upcoming exascale systems in terms of *power* and *energy* efficiency.

Keywords Block size · Exascale computing · GPU · Power and energy

1 Introduction

The *exascale* systems (the next generation of high-performance computing (HPC)) will provide unprecedented processing power and memory so that researchers can perform very large-scale simulations. There are many obstacles to reaching exascale performance (order 10^{18} floating-point operations per second) such as the increasing power and energy, more fault rate, and updating the applications from petascale (the current HPC Systems) to exascale computing (the next-generation HPC systems) [8, 9, 43, 46]. Among these, power consumption

✉ Muhammad Jawad Ikram
m.jawad@jicollege.edu.sa

[1] Jeddah International College, Jeddah 23831, Saudi Arabia

[2] King Abdulaziz University, Jeddah 21589, Saudi Arabia

is the main hurdle to achieving exascale performance [43]. Innovations and improvements in many areas such as algorithms, architecture, and software are required to reach the exascale performance, as the current HPC techniques are not suitable for the upcoming exascale systems [8]. Alternative solutions, which can survive with energy consumption limitations, are needed to make exascale performance a reality.

During the last decade, graphics processing units (GPUs) have become a promising architecture in the pursuit of exascale systems due to advancements in *power* and *performance* efficiency. For that reason, we further explored GPUs to provide insights for exascale systems in terms of *power* and *energy* efficiency. In our experimental study, NVIDIA Tesla K40 [30] GPU was used, which provides outstanding high-performance capabilities with *power* efficiency. Tesla K40 is specifically manufactured by NVIDIA Corporation to give solutions for the most overwhelming supercomputing challenges [30]. NVIDIA Corporation also designed other GPUs (Kepler architecture) that include Tesla K20 GPU and Tesla K80 GPU. NVIDIA Kepler architecture is utilized by numerous scientific applications [37] because of its ground-breaking computing technology. Apart from this, Tesla K40 is adopted by a large number of top500 [49] and green500 [50] supercomputers, as it provides significant supercomputing capabilities. In this study, we used the Compute Unified Device Architecture (CUDA) [33] as a programming platform on NVIDIA Tesla K40 GPU. Some CUDA terminologies related to our experiments are briefly described below, as stated in [33]:

- *Host:* means the processor and its memory. In this study, it is the Intel(R) Xeon(R) CPU E5-2640 2.50 GHz and its memory.
- *Device:* means the graphics processing unit and its memory. In our case, it is the K40 GPU and its memory.
- *Kernel:* is a function that runs on the device. Programs are executed in parallel on GPU as kernels. Various *threads* are used to launch a single kernel at a time. In this study, we have *Bitonic Mergesort (BM)* [1] and *Vector Addition (VA)* [35] kernels, which contain the relevant source codes.
- *Wrap:* is a collection of 32 successively numbered *threads* inside a single *thread block*.
- *Block Size (threads per block)* and *Grid Size (blocks per grid):* A CUDA program executes parallel functions on the GPU known as *kernels* that run across many parallel *threads*. The programmer or compiler arranges these *threads* into *thread blocks* and *grids of thread blocks,* as described in Figs. 1 and 2.
- *Kernel Configuration:* is defined by *block size (threads per block)* and *grid size (blocks per grid)*. It is calculated based on the product of *block size* and *grid size*. It needs to be defined while launching the kernel on the GPU. The maximum *block size* for a one-dimensional *kernel launch* on NVIDIA Tesla K40 GPU is 1024.
- *Kernel Launch:* Triple angle brackets mark a *kernel launch* from *host* code to *device* code. For example, "*mykernel*<<<*blocks per grid threads per block*>>>()." Each *thread* that executes the *kernel* is given a unique *thread ID* that is accessible within the *kernel* through the built-in *threadIdx* variable. Paral-

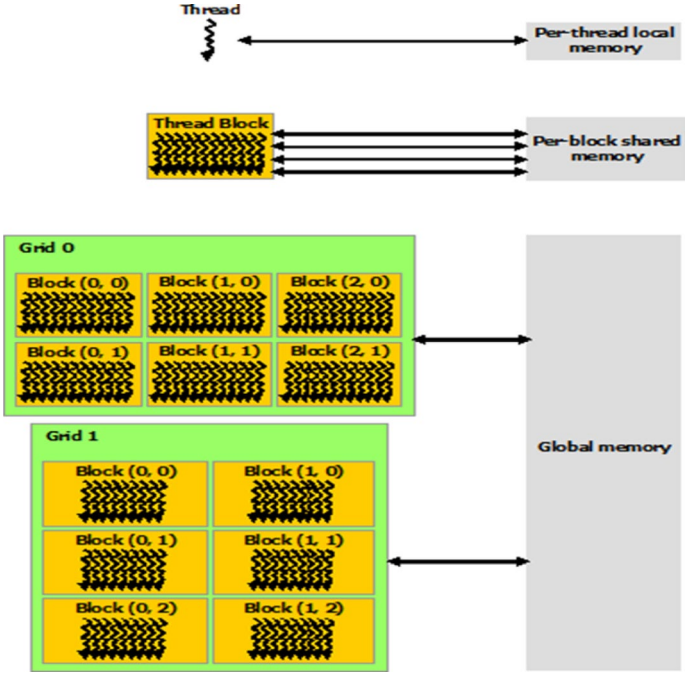

Fig. 1 CUDA hierarchy of threads, blocks, and grids, with corresponding per-thread private, per-block shared, and per-application global memory spaces [33]

lel *threads* can be launched using the following way. Launch N *blocks per grid* with M *threads per block* with kernel$<<<N, M>>>()$.

In our recent studies [3] and [5], it was observed that investigating *power* and *energy* consumption of basic software building blocks on GPUs, such as sorting algorithms, can provide a new perspective to reduce the power requirement of the upcoming exascale systems. It was observed that depending on the algorithm's fundamental design, some algorithms provide an inherent power and energy saving as compared to other algorithms [3] and [5]. In [3] and [5], it was found that a simple Bitonic Mergesort (BM) algorithm [1] was more *power*- and *energy-efficient* than a performance-optimized *Advanced Quicksort (AQ)* algorithm [34], while compared under various workloads on NVIDIA Tesla K40 GPU. Furthermore, it was suggested that a significant amount of *power* and *energy* saving can be achieved based on the selection of *power*- and *energy-efficient* algorithms. We also identified some programming-related factors that affect GPU power consumption. In this research, we suggest that further *power* and/or *energy saving* can be achieved by selecting the proper *block size* in *kernel configuration* while executing the workload. In order to study the effect of varying *block size* in GPU kernels, we executed BM [1] and VA [35] kernels on a dataset of 512 M elements (unsigned integer random numbers) on NVIDIA Tesla K40 GPU. For random number generation, we used the C

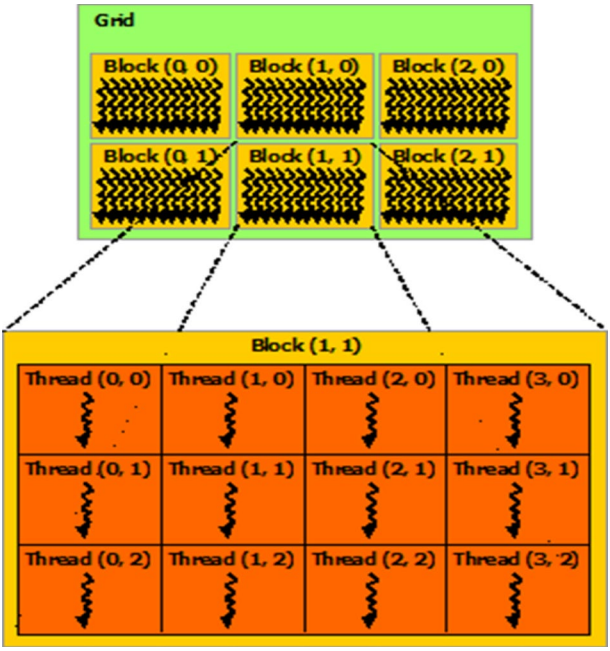

Fig. 2 CUDA hierarchy of threads, blocks, and grids, with corresponding per-thread private, per-block shared, and per-application global memory spaces [33]

built-in method rand() (method for generating pseudo-random numbers) and constant RAND MAX (greatest value returned by the rand() method).

Next, we discuss the two GPU kernels, namely BM [1] and VA [35] that were used in the experimental analysis. Firstly, we performed the experiments on BM and collected the results. Then, to further validate our findings, we performed the same experiments on another GPU kernel, i.e., VA kernel. BM is an example of data-independent algorithms that was developed by Ken Batcher [7]. BM was designed specifically for parallel architectures. It can work as a building method for developing sorting networks that contain $O(n(\log^2(n))$ number of comparators and consist of $O(n(\log^2(n))$ delay, where n shows the length of the list to be sorted [2]. BM is suitable for single instruction, multiple data platforms because it can work in-place and with minor interprocess communication. In this study, we used the CUDA version of BM that is based on [1]. The other kernel that we used in experiments is a simple vector addition program available in CUDA SDK [35]. This program uses two vectors A and B with dimensions *(wA, hA)* and *(wB, wA)*, respectively, and then divides the task among various threads to get the product C of vectors A and B.

We used *peak power, average power, energy,* and *kernel runtime* to study the *power* and *energy* efficiency of the BM and VA kernels with varying *block sizes* [3, 5], and [21]. Furthermore, we studied some GPU performance counters, i.e., *achieved occupancy* and *eligible warps* metrics (based on NVIDIA Nsight Visual Studio Edition 5.2 [36]), to know why different GPU kernels exhibit different power and energy consumption for similar workloads while varying the *block size*

in the *kernel configuration*. It is pertinent to mention that performance counters have a relation with the *power* and *energy* consumption of kernels executing on the GPU, as stated by Luo and Suda [27], "Even with a powerful hardware in parallel execution, it is still difficult to improve the application performance and reduce energy consumption without realizing the performance bottlenecks of parallel programs on GPU architectures." The power, energy, and performance metrics are described below, as stated in [3]:

- *"Peak Power:* is the peak level of GPU *power that* is reached when the kernel is executing on the GPU. The *power profile* indicates its value.
- *Average Power:* is the *average power* of a kernel executing on the GPU. It is obtained from the power profile of the kernel by adding all the sampled power values and dividing them by the total number of obtained power samples.
- *Energy:* is the total *energy* consumed by the kernel executing on the GPU. The area under the *power curve* of a kernel indicates the total energy consumed during execution. This metric is calculated by integrating the power curve over kernel runtime.
- *Kernel Runtime:* is the runtime of a kernel that is executing on the GPU. During this time, the GPU consumes more power and energy than its *idle state power*.
- *Achieved Occupancy:* is the ratio of *active warps* in each *streaming multiprocessor (SM)* to the maximum possible *active warps* (on a K40 SM, maximum *active warps* = 64) [25]. A *warp becomes* active from the time its threads begin execution on the SM to the time it completes the last instruction. The following equation describes the *achieved occupancy:*

$$\text{Achieved Occupancy} = \frac{\text{active warps}}{\text{maximum active warps}} \tag{1}$$

- *Eligible Warps:* shows the number of cycles that a *warp scheduler* [30] had at least one *eligible warp* to select from. The higher the percentage of cycles with *eligible warps*, the more efficient the code runs on the target device. A streaming multiprocessor (SM) has one or more warp schedulers, each of which tries to issues instructions from a *warp* on each clock cycle. To sufficiently hide latencies between dependent instructions, each warp scheduler must have at least one *eligible warp* to issue an instruction every clock cycle. Keeping the *achieved occupancy* high during *kernel runtime* avoids situations where all *warps* are stalled and no instructions are issued [36]."

Generally, this research has the following contributions:

- The research highlights insights for exascale systems by providing new ways to achieve power and/or energy efficiency.
- The research demonstrates how power and/or energy saving can be achieved by varying the *block size* in kernel configuration while executing the workload on GPU.

- This paper presents a novel idea of saving power and energy by varying *block size* in a *kernel configuration*.

The remaining paper is structured as follows. Section 2 presents the related literature review. Section 3 briefly describes the experimental setup. In Sect. 4, we discuss the results. Section 5 concludes the work and identifies some future prospects from this work.

2 Literature review

Vila et al. [52] presented the study carried out by NVIDIA Corporation regarding the design of exascale systems by incorporating the properties that answer the scaling problems of performance and energy efficiency. The researchers highlighted that more innovations in designing algorithms and architecture are required in order to bring improvements in the performance of applications for addressing memory locality, the scaling issue, and integer execution efficiency.

Dally [18] firstly demonstrated the hurdles of the upcoming exascale systems and then discussed the recent techniques to overcome those hurdles. The researcher argued that an improvement of 200-fold will be essential in energy per instruction to reach the expected exascale performance in a reasonable power budget, i.e., below 20 Megawatts. Moreover, he argued that a number of creative programming environments will also be essential for exascale systems' programming.

Zhao and Chen [56] suggested a GPU general-purpose computing recent prediction on the basis of analyzing various GPU architectures and rules of CUDA program execution. They demonstrated that the prediction error of a single program was less than 10%, and the average prediction error was less than 6. Obrien et al. [38] surveyed power and energy predictive models in high-performance computing applications and systems. They emphasized the shortcomings of the power and energy efficiency models to accurately predict the power and energy consumptions keeping in view the heterogeneous behavior of closely integrated high-performance computing systems. Furthermore, Bridges et al. [11] surveyed GPU power modeling and profiling techniques. They focused on the use of GPU built-in (the internal sensor) and external sensors for measuring GPU power consumption. They also highlighted the improvements and hurdles of counter-based GPU power modeling.

Vijaykrishnan et al. [53] examined the power consumption of applications on GPUs and other heterogeneous platforms. Furthermore, Nagasaka et al. [25] presented statistical models for GPUs on the basis of performance counters provided by vendors. Furthermore, Kasichayanula et al. [23]. showed a study of various microbenchmarks running on GPUs from a perspective of power consumption.

Suda and Ren [47] presented two models for the prediction of runtime and energy consumption. Their models help programmers to understand the performance and energy-saving bottleneck of parallel applications on GPUs. Ren and Suda [48] suggested a load sharing technique for adjusting the workload assignment within the CPU and GPU components inside a CUDA processing element (PE) with the objective of overall power optimization.

Lim et al. [26] proposed a power model on the basis of McPAT [26] for GPUs. Their model used initial data from McPAT that contained detailed GPU configuration. Subsequently, they adjusted the model by comparing it with empirical data. They used NVIDIAs Fermi architecture for developing the power model. Chen et al. [13] suggested an integrated power and performance (IPP) prediction model for GPU architecture. Their model is able to predict the optimal number of active processors for a given application. The researchers argued that application performance cannot be improved when applications reach the peak memory bandwidth and utilize more cores. Unlike previous models, their model is able to predict the dynamic events happening on the GPU.

Zhang et al. [55] utilized a tree-based random forest technique for the development of a power consumption model with improved accuracy, as compared to regression-based methods. They used the model to investigate the correlation among individual performance metrics. Likewise, Pool et al. [42] used random forest techniques for studying the power consumption and performance of ATI GPUs. Alternatively, Lee et al. [25] developed a power and energy model on the basis of energy consumed per unit by each instruction. Roy et al. [45] argued on measurement of energy as a foundation for algorithm design and implementation. They observed that memory parallelism has a relationship with the energy consumption of algorithms. The researchers used an asymptotic energy complexity model [44] for validation. They also observed that a significant energy saving in quicksort and mergesort can be attained by varying the parallelization.

Padoin et al. [41] examined iterative applications and studied the load balancing thresholds for energy saving on those applications. They suggested two new versions of energy-aware load balancer with the objective to decrease the energy consumption of parallel architectures that execute imbalanced scientific applications with no loss in performance.

Zecena et al. [54] investigated performance and energy study of both serial as well as parallel odd–even sort, shell sort, and quicksort on a shared memory system comprising of two quad-core AMD 2380 Opteron processors. The researchers used iterative odd–even sort and shell sort algorithms. On the other hand, they used a recursive version of quicksort.

Ukidave et al. [51] studied various methods of optimization for power and performance efficiency on a number of heterogeneous platforms that include discrete GPUs, shared memory GPUs, low power system-on-chip devices. They also used hardware platforms from NVIDIA, Intel, and Qualcomm Corporations. The researchers studied the energy efficiency of optimization techniques for identifying the power-performance trade-off. They found that architectural and algorithmic factors have an effect on power consumption. They presented that algorithms implementing the same operation may have different performances depending on application design and target hardware. Similarly, Padoin et al. [40] evaluated the performance and energy efficiency of applications on hybrid CPU and GPU architecture. They used a scientific application from the area of agroforestry as a sample and examined how the application workload may affect the energy consumption on hybrid architectures.

Al-Hashimi et al. [4] investigated power and energy consumption of three control loops (*for, while,* and *do-while*) on Intel Sandy Bridge CPU. Their study found that for loop can provide power saving as compared to the other two loops. Al-Hashimi et al. [5] and Aljabri et al. [6] studied power and energy consumption of generic mergesort and an optimized quicksort on different Intel Architectures, i.e., Sandy Bridge and Ivy Bridge, respectively. They found that mergesort can provide power saving as compared to quicksort while executing a similar workload. Similarly, Jamal et al. [22] (our new research) studied the power efficiency of matrix multiplication algorithms in high-performance computing. Their results suggest that power efficiency can be improved by selection of algorithms. In addition, Dlamini et al. [19] performed a meta-analysis of two quicksort and merge sort algorithms. They concentrated on energy consumption of embedded systems and mobile devices. Likewise, Boughzala et al. [10] analyzed CUDA kernels by predicting the energy consumption through simulation. The researcher claim that their model can be used by users to achieve energy efficiency in general-purpose GPU applications. In the same way, Kondo et al. [24] discussed the development of a software framework for optimization of code and management of system power for post-petascale supercomputers. Montana and Cheptsov [28] considered achieving energy efficiency by logging the energy consumption of different algorithms, and then selecting the more energy-efficient algorithm for fulfilling requirements of an application.

Lee et al. [25] used value similarity to improve the power consumption of a register file. They presented a warp compression scheme for GPU register files based on similarity of register values that resulted in decreasing both dynamic and leakage power.

Connors and Qasem [14] investigated the effect of thread *block size* and register allocation on the performance and power efficiency of three heuristic search algorithms. They found a reasonable variation in the performance of codes that were executed with identical occupancy levels. They found that the highest occupancy did not always result in the best power-performance trade-offs. They also showed that, for a given occupancy level, having larger thread blocks, and accordingly fewer thread blocks per SM, usually results in improving both power and performance.

Coplin and Burtscher [17] investigated source code optimization and its outcome on GPU performance, power draw, and energy efficiency. They examined 128 versions of two n-body codes and evaluated the active execution time and the power consumption of each code version on three inputs, various GPU clock frequencies, two arithmetic precisions, and with and without ECC. Most of the previous works rely on using external power meters, and statistical and estimation models for power and energy investigations. Recent high-performance GPUs come with an on-board sensor that can be queried dynamically to collect power-related data from the GPU board. Burtscher et al. [12] used the GPU onboard sensor for power and energy measurements on NVIDIA Kepler K20 GPU. Their work identified various anomalies in power and energy measurements. In addition, Coplin and Burtcher [16] studied the power and energy consumption of irregular and regular programs executing on K20 GPU using the onboard GPU sensor. Furthermore, Coplin and Burtscher [15] investigated the energy efficiency, power draw, and runtime of 34 programs from 5 general-purpose GPU benchmark suites. Likewise, Ferro et al. [20]

investigated the GPU sensors to obtain high-resolution power profiles of real and benchmark applications. They developed their own tools for querying the sensors of two different models of NVIDIA GPUs in order to compare their accuracy.

Exploiting the GPU built-in sensor, Ikram et al. [21] (our previous work) proposed an experimental approach for measuring power and energy consumption of GPU kernels on NVIDIA Kepler GPUs using the GPU on-board sensor. The experimental approach can help us obtain the power profile of any program executing on Kepler GPUs. Using the power profile, we can investigate the *peak power, energy,* and *kernel runtime* of the kernel executing on the GPU. This experimental approach was used by Al-Hashimi et al. [5] (our previous work) to study the power and energy consumption of Bitonic Mergesort (BM) and Advanced Quicksort (AQ) on NVIDIA Tesla K40 GPU. They observed that in most cases, a simple BM offers a considerable power and energy advantage over an optimized quicksort algorithm. Furthermore, Abulnaja et al. [3] (our previous work) investigated power and energy consumption of BM and AQ based on performance evaluation. The study identified that some algorithms (such as BM) have inherent power and energy advantage over other algorithms while executing the same workload, which can offer new ways to address the power obstacle of exascale systems.

Inspired by the observations in the above studies, this paper takes an orthogonal approach to investigate power and/or energy-saving techniques that can be fruitful for the exascale systems. In this regard, we propose how power and/or energy saving can be achieved by varying the *block size* in *kernel configuration* while executing the workload on GPU.

3 Experimental setup

The experimental study was conducted on a Fujitsu HPC workstation (system specification given in Table 1) with a dedicated NVIDIA Tesla K40 GPU as a test platform. We used NVIDIA System Management Interface (NVSMI) [32] to read the GPU sensor for obtaining the power profile of the kernel executing on the GPU. NVSMI is a cross-platform utility that can provide dynamic monitoring and management of activities occurring on NVIDIA GPUs. NVSMI is based on C-based NVIDIA Management Library (NVML) [31]. Data from NVSMI are recorded into a log file, which contains *NVSMI Logs* recorded at a sampling rate of 66.6 Hz. The information includes *power management data* and *performance states* with proper *timestamps* as explained in [3, 5, 21], and [32].

Table 1 System specifications

Manufacture	Model	CPU	Menory
Fujitsu	CELSIUS M720 POWER	Intel(R) Xeon(R) CPU E5-2640 2.50 GHZ, 2494 MHZ & 8.00 GB	8.0 GB

The Visual Studio 2013 compiler configured in *release mode* and $\times 64$ active solution platform was used for executing the code on NVIDIA Tesla K40 GPU. In order to leverage the parallel programming capabilities offered by NVIDIA GPUs, we used the Compute Unified Device Architecture (CUDA) [33] as a programming platform in our experiments. Apart from NVSMI and CUDA, we used Origin 6.0 [39] (a tool for mathematical and statistical analysis) and MS Excel Worksheets to accomplish experiments effectively on Tesla K40 GPU.

We used a dataset of 512 M elements in our experiments as expressed in Table 2. In order to execute each element of the input array on a different thread for parallel execution on the GPU, we need to launch a kernel with 512 M threads. For that reason, we selected different *block sizes* starting from 1 (the minimum on NVIDIA K40) and going till 1024 (the maximum on NVIDIA K40). It should be noted that an optimum *block size* must be a power of 2. Thus, for a kernel configuration with 512 M threads, we changed both the *block size* and *grid size*, as expressed in Table 2.

We used the experimental setup described in [21] with some modifications, for investigating the effect of *block size* on GPU kernel power and energy consumption. The experimental setup is briefly described here using Fig. 3, which shows the full power profile (idle + active) of GPU while executing a BM Kernel on a dataset of one Giga elements. The power profile illustrates the current *power draw* of the NVIDIA Tesla K40 GPU board that was monitored through the built-in GPU sensor at a sampling frequency of 66.6 Hz (as recommended in [12]) through NVSMI [32]. Idle and active states of Tesla K40 GPU, *kernel runtime,* and *peak power* are highlighted in Fig. 3, i.e., point "A" shows the active state starting point (at this point the GPU power level starts increasing), point "B" identifies the GPU *peak power,* and point "C" shows the end of the GPU active state, as the GPU power level reaches the idle state power (20.57 W for NVIDIA Tesla K40 GPU [30]) again.

The idea of this research is to explore the effect of varying block size on GPU power and energy consumption, and to provide insights for next-generation high-performance computing (exascale) by providing new ways to achieve power and energy efficiency. For this purpose, we selected two fundamental building blocks of various high-performance computing applications that are sorting algorithms and vector addition.

Table 2 Kernel size for 512 M elements with varying block size and grid size

Block size	Grid size	Kernel size
1	536,870,912	536,870,912
64	8,388,608	536,870,912
128	4,194,304	536,870,912
256	2,097,152	536,870,912
512	1,048,576	536,870,912
1024	524,288	536,870,912

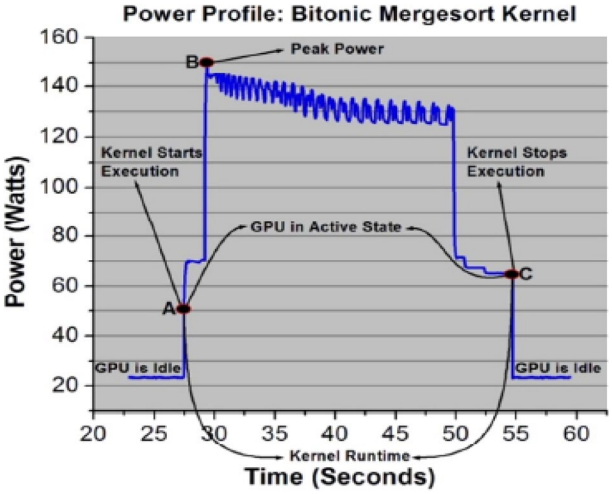

Fig. 3 Full power profile of GPU (idle state + active state) for BM: Dataset = 1G elements [3]

The stepwise description of the experimental procedure for executing BM or VA on a dataset of 512 (536,870,912) Mega (M) elements is below:

1. Invoke NVIDIA System Management Interface (NVSMI) from command prompt to read on-board GPU sensor; create a log file; and generate a query to record *power readings, performance states,* and *timestamps* at a sampling rate of 66.6 Hz (every 15 ms). This sampling rate (66.6 Hz) results in accurate power measurements as recommended in [12].
2. A dataset of 512 M unsigned integer random numbers is generated at runtime in the source code using rand() % RAND MAX.
3. A GPU kernel, i.e., Bitonic Mergesort (BM) or Vector Addition (VA), is launched on the dataset in the source code with varying *block sizes* in the *kernel configuration* based on Table 2. For every *block size,* the *kernel configuration* is such that each number in the input array is allocated one thread for parallel execution on the GPU.
4. As the source code gets executed, stop NVSMI execution in the command prompt.
5. Extract data from *NVSMI log files to* MS Excel Worksheet and filter data for *power draw, performance states,* and *timestamps.*
6. Extract *power* and *timestamps* values only for *P0* performance state (GPU active state) from MS Excel Worksheet to a mathematical package, i.e., Origin 6.0 [39].
7. Find the *kernel runtime* from the power profile and *timestamps* values using the following equation based on Fig. 3:

$$\text{Kernel Runtime} = t_C - t_A \tag{2}$$

where t_A and t_C represent the timestamps of the power profile at Point A and Point C, respectively.

8. Obtain the power profile of the dataset over *kernel runtime* and record the *peak power.* Based on Fig. 3, the *peak power* is described by the following equation: Obtain power profile of the dataset over *kernel runtime* and record the *peak power.* Based on Fig. 3, the *peak power* is described by the following equation:

$$\text{Peak Power} = \max\left(P(t)\right) = P_{t_B} \tag{3}$$

where $P\ (t)$ indicates the power profile of the kernel that is executing on the GPU, t_B represents the *timestamps* of the power profile at Point B, and Pt_B indicates the *peak power*, which represents the maximum value in the power profile, as shown in Figure 3.

9. Obtain the *average power* from the power profile based on the following equation based on Fig. 3:

$$\text{Average Power} = \frac{\sum_{t=t_A}^{t_C} P(t)}{\text{number of samples}} \tag{4}$$

10. Using Origin 6.0, integrate power curve to obtain *energy* consumption of the kernel using the following equation based on Fig. 3:

$$\text{Energy} = \int_{t_A}^{t_C} P(t) \tag{5}$$

where $t_A \leq \mathrm{t} \leq t_C$

11. Obtain *achieved occupancy* and *eligible warps* using NVIDIA Nsight Visual Studio Edition 5.2 [36].
12. Repeat step 1 to step 11 10 times to compute average values for *peak power, average power, energy, kernel runtime, achieved occupancy,* and *eligible warps.*

We first executed BM 10 times based on the above experimental procedure and collected the average results for each metric. In order to further validate the results, we conducted the same experiment on another kernel, i.e., VA kernel. Identical observations were derived from both the experiments (explained in Sect. 4.)

4 Results evaluation

Figure 4 shows the *average power* and *peak power* consumption of Bitonic Mergesort (BM) versus various *block sizes*. The results suggest that for smaller *block sizes* (*block size* < 64), the GPU *average power* and *peak power* consumption are low. As we increase the *block size* (*block size* > 64), the *peak power* and *average power* also get increase and remain almost at the same level for *block size* between 128 and 1024. It is pertinent to mention that lower *peak power* and *average power* at lower *block size* (*block size* < 128) are due to the lower *achieved*

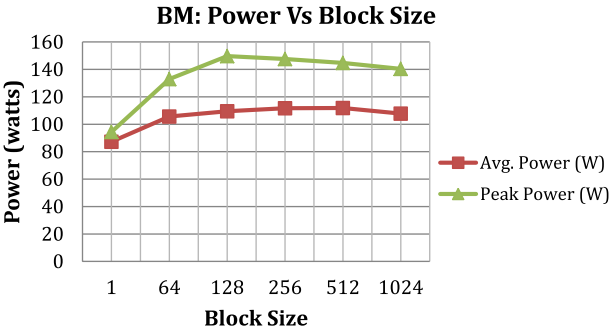

Fig. 4 BM: power vs block size

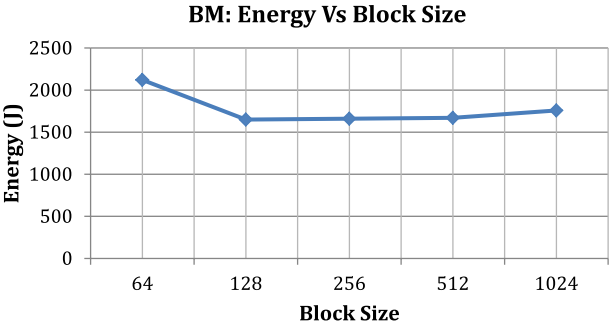

Fig. 5 BM: energy vs block size

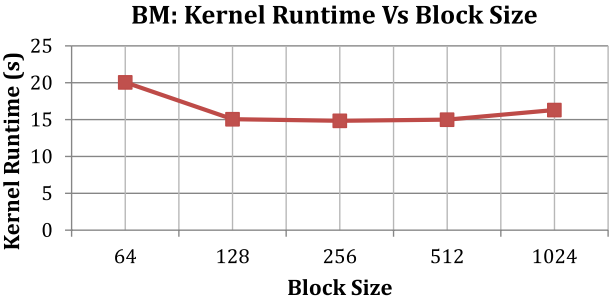

Fig. 6 BM: kernel runtime vs block size

occupancy and *eligible warps* across the streaming multiprocessors of the Tesla K40 GPU. As identified in [3] that occupancy has a relation with both performance and power consumption of the kernel executing on the GPU. Figures 5 and 6 show the energy consumption and kernel *runtime of* BM with varying *block sizes*, i.e., 64 to 1024, respectively. We have not shown results of *energy* and *kernel runtime* for 1 *block size*, as both these metrics have very large values, i.e.,

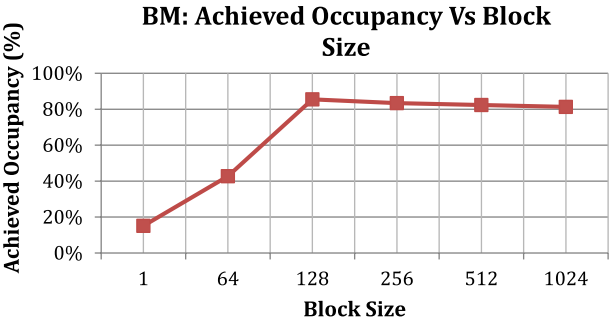

Fig. 7 BM: achieved occupancy vs block size

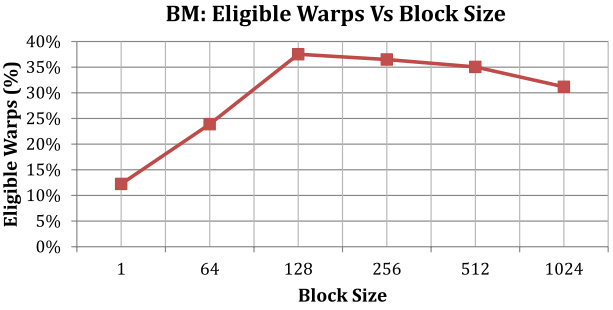

Fig. 8 BM: eligible warps vs block size

68,475.97 J and 784.48 s, respectively. The energy consumption of BM is very high because of higher *kernel runtime*, lower *achieved occupancy*, and lower *eligible warps* while executing with a *block size* of 1. The results of *kernel runtime* for BM are shown in Fig. 6. The results reveal that lower *block size* (*block size* < 64) consumes more energy but has the benefit of lower *peak power* and *average power* consumption.

Next, we discuss why different *block sizes* have different *power, energy,* and *kernel runtime?* To know the answer, we studied some performance counters, i.e., *achieved occupancy* and *warp issue efficiency* as discussed in Sect. 1. Figures 7 and 8 show the results of BM *achieved occupancy* and *eligible warps* versus various *block sizes*, respectively. Figure 7 suggests that the achieved occupancy is very low for *block size* 1 and 64, i.e., 15.12 and 42.75%, respectively, while for larger *block size* (*block size* > 64), we have adequate *achieved occupancy*. For a *block size* of 128, we have the highest *achieved occupancy* and *eligible warps*, i.e., 85.50 and 37.52%, respectively. A further increase in the *achieved occupancy* leads to an undesirable effect on the kernel performance due to reduction in resources per thread [36]. On the other hand, lower *achieved occupancy* leads to poor instruction issue efficiency because of fewer *eligible warps* to hide latency between dependent instructions [3]. As shown in Fig. 8, for lower *block size* (*blocksize* < 128), we have fewer eligible *warps*. This is because of lower *achieved occupancy* at smaller *block sizes*. As the

achieved occupancy gets higher (*block size* > 64), the percentage of *eligible warps* also gets higher.

Achieved occupancy and *eligible warps* have also an effect on *kernel runtime* and *energy* consumption of the kernels. As shown in Figs. 7 and 8, the highest *achieved occupancy* and *eligible warps are* at a *block size of* 128. If we observe Figs. 5 and 6, we can find that the minimum *energy* and *kernel runtime* are also at a *block size* of 128. Investigating *achieved occupancy* and observing its effect on *kernel runtime, power,* and *energy* when running at different occupancy levels, should be considered as one of the first levels triages in examining kernels' performance on the device. Thus, a desirable *block size* can be chosen based on power or energy saving. In this particular case, if we want energy saving, we should select a *block size* of 128. On the other hand, if we want power saving, we should select a *block size* less than 128. Accordingly, if both power and energy saving are required, then a trade-off between power and energy should be identified. The trade-off should be an optimal setting (means *block size*) for both power and energy. For instance, in the case of BM, if both power and energy saving are required, an optimal *block size* should be 1024 because, at this *block size,* BM is consuming 140.39 watts of power, 1758.37 J of energy, and *kernel runtime* of 16.30 s. This is a trade-off setting for BM under which power saving is achieved at the cost of a slight increase in energy and *kernel runtime*. In summary, the results suggest that *block size* have a significant effect on the *power, energy,* and *kernel runtime* of the code executing on the GPU, and properly selecting *block size* result in both power and energy saving whichever is desirable.

In order to further validate the above results and observations, we repeated the experiments on another kernel, i.e., vector addition (VA) kernel, as shown in Figs. 9, 10, 11, 12, 13. We found a similar trend as in the case of BM. For example, in the case of VA kernel, for smaller *block size* (*block size* < 128), we have lower *peak power* and *average power*. Similarly, as in the case of BM, here again, *energy* and *kernel runtime* is very high for smaller *block size* (*block size* < 128). More interestingly, Figs. 10 and 11 suggest that the VA kernel has the minimum energy and kernel runtime at a block size of 128, as was the case for the BM kernel. The results also suggest that higher achieved occupancy can be attained

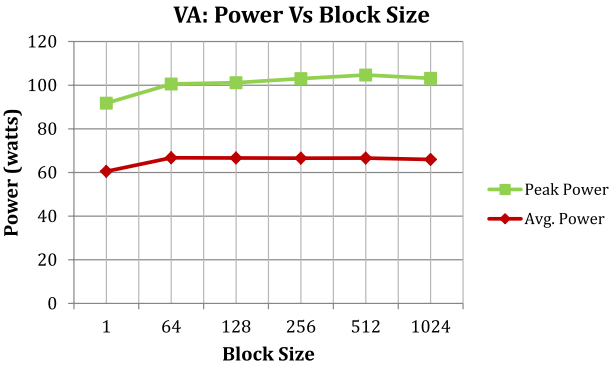

Fig. 9 VA: power vs block size

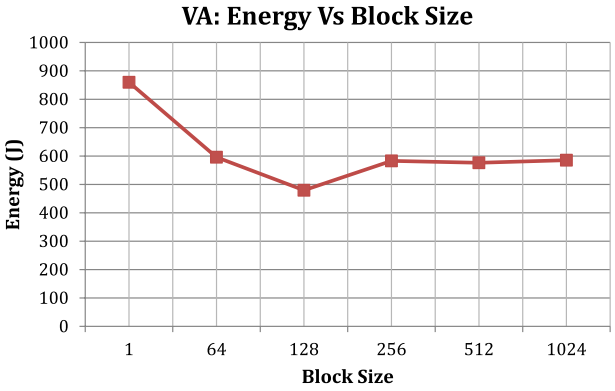

Fig. 10 VA: energy vs block size

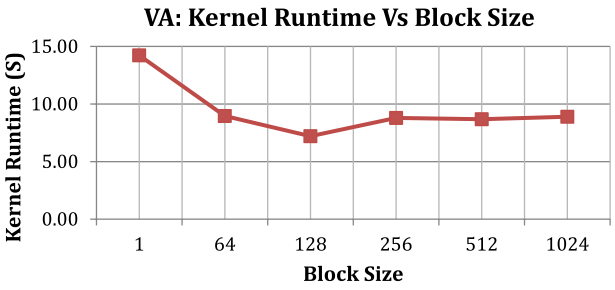

Fig. 11 VA: kernel runtime vs block size

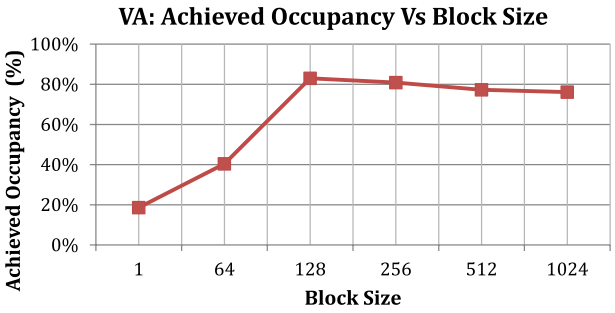

Fig. 12 VA: achieved occupancy vs block size

by selecting *block size* greater than 64 both in the case of BM and VA kernels, as shown in Figs. 7 and 12, respectively. The same trend is also observed for *eligible warps* metrics for both BM and VA kernels, as shown in Figs. 8 and 13, respectiv ely.

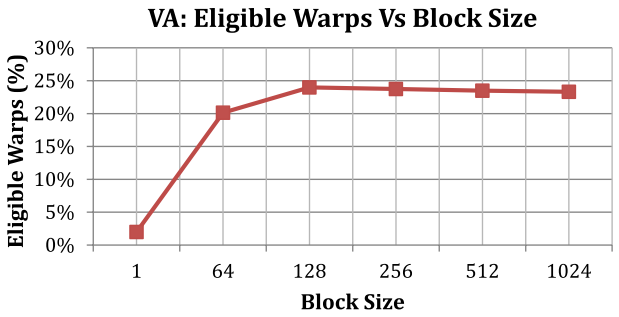

Fig. 13 VA: eligible warps vs block size

Table 3 BM kernel results with varying block size

Block size	Average power (W)	Peak power (W)	Energy (J)	Kernel runtime (S)	Achieved occupancy (%)	Eligible warps (%)
1	87.29	94.35	68,475.97	784.8	15.12	12.23
64	105.68	132.92	2120.92	20.05	42.75	23.89
128	109.49	149.65	1649.97	15.06	85.50	37.52
256	111.66	147.55	1660.11	14.85	83,039	36.49
512	111.83	144.71	1670.28	15	82.36	35.06
1024	107.77	140.39	1758.37	16.3	81.33	31.18

Table 4 VA kernel results with varying block size

Block size	Average power (W)	Peak power (W)	Energy (J)	Kernel runtime (S)	Achieved occupancy (%)	Eligible warps (%)
1	60.51	91.79	859.91	14.23	18.61	1.98
64	66.72	100.56	595.99	8.96	40.41	20.13
128	66.66	101.13	479.27	7.21	83.00	23.98
256	66.56	103.02	583.04	8.79	80.88	23.74
512	66.6	104.65	576.37	8.68	77.27	23.48
1024	65.96	103.15	585.42	8.9	76.13	23.32

The results further suggest that in a high-performance computing environment, a reasonable power or energy saving (whichever is desirable) can be achieved by selecting a proper *block size* in *kernel configuration* while executing the code on the GPU. The results of BM and VA kernels are summarized in Tables 3 and 4.

5 Conclusions

This study identifies that *block size* in a *kernel configuration* has a significant effect on the power and energy consumption of the kernel executing on the GPU. Selecting *kernel configuration* with power-optimum, energy-optimum, or both power- and-energy-optimum *block size* can offer a reasonable amount of power and/or energy saving for kernels executing on GPUs. The exascale research community can benefit from the obtained results in selecting the optimum block size for applications executing on GPUs in order to reduce the excessive power requirements of exascale systems. Two kernels (BM and VA kernels) were investigated to study the effect of varying *block size in kernel configuration* on GPU power and energy consumption. In general, the following are some key observations from this study:

- Changes in the *block size (threads per block)* significantly affect the power, energy, and performance efficiency of the kernel executing on the GPU. For the prospective exascale computing environment, investigating power and energy consumption with varying *block size* should also be taken into consideration along with *achieved occupancy* as first-level triage to determine the power, energy, and performance efficiency of kernels executing on GPUs.
- Lower *block size* generally results in lower *achieved occupancy* and lower *eligible warps* across all the streaming multiprocessors (SMs). Having lower *achieved occupancy* and *eligible warps* across all the SMs limit the warp scheduler's ability to make forward progress that results in higher *kernel runtime* and higher energy consumption of the algorithm (even with significantly lower power consumption).
- Keeping the *block size very* small (*block size* < 64) results in very high *kernel runtime* and *energy* but on the other hand, lower *block size* has an advantage of lower power consumption. So, if both power and energy saving are required, then a trade-off between power and energy should be identified that should be an optimal setting (means *block size*) under which both power and energy are at an acceptable range.

As future work, it would be interesting to repeat these experiments on other heterogeneous architectures and investigate some other fundamental algorithms along with BM and VA kernels. Additionally, it would also be interesting to investigate how the system will react in the concurrent execution mode with respect to variation in *kernel configuration.* Furthermore, it would also be attracting to add some optimization algorithms and develop a unique formula to get the best *block size* and *grid size*.

Acknowledgements The authors are thankful to the Deanship of Scientific Research, King Abdulaziz University, for supporting this research.

Funding This work was supported by the Deanship of Scientific Research, King Abdulaziz University, under Grant 1–611-1433/HiCi.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. (date accessed February 10, 2019) Parallel Bitonic Mergesort. URL http://www.tools-of-computing.com/tc/CS/Sorts/bitonic_sort.htm
2. (date accessed February 11, 2019) Bitonic Sort. URL http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/oddn.htm
3. Abulnaja OA, Ikram MJ, Al-Hashimi MA, Saleh ME (2018) Analyzing power and energy efficiency of bitonic mergesort based on performance evaluation. IEEE Access 6:42757–42774
4. Al-Hashimi M, Saleh M, Abulnaja O, Aljabri N (2014) Evaluation of control loop statements power efficiency: An experimental study. In: 2014 9th International Conference on Informatics and Systems, IEEE, pp 45–48
5. Al-Hashimi MA, Abulnaja OA, Saleh ME, Ikram MJ (2017) Evaluating power and energy efficiency of bitonic mergesort on graphics processing unit. IEEE Access 5:16429–16440
6. Aljabri N, Al-Hashimi M, Saleh M, Abulnaja O (2019) Investigating power efficiency of mergesort. J Supercomput 75(10):6277–6302
7. Batcher KE (1968) Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, spring joint computer conference, pp 307–314
8. Beckman P (2011) On the road to exascale. Sci Comput World 116:26–28
9. Bergman K, Borkar S, Campbell D, Carlson W, Dally W, Denneau M, Franzon P, Harrod W, Hill K, Hiller J, et al. (2008) Exascale computing study: Technology challenges in achieving exascale systems. Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech Rep 15
10. Boughzala D, Lef_evre L, Orgerie AC (2020) Predicting the energy consumption of cuda kernels using simgrid. In: 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), IEEE, pp 191–198
11. Bridges RA, Imam N, Mintz TM (2016) Understanding gpu power: a survey of profiling, modeling, and simulation methods. ACM Comput Surv (CSUR) 49(3):1–27
12. Burtscher M, Zecena I, Zong Z (2014) Measuring gpu power with the k20 built-in sensor. In: Proceedings of Workshop on General Purpose Processing Using GPUs, pp 28–36
13. Chen J, Li B, Zhang Y, Peng L, Peir Jk (2011) Tree structured analysis on gpu power study. In: 2011 IEEE 29th International Conference on Computer Design (ICCD), IEEE, pp 57–64
14. Connors T, Qasem A (2015) Power-performance analysis of metaheuristic search algorithms on the gpu. In: 2015 Sixth International Green and Sustainable Computing Conference (IGSC), IEEE, pp 1–6
15. Coplin J, Burtscher M (2015) Effects of source-code optimizations on gpu performance and energy consumption. In: Proceedings of the 8th Workshop on General Purpose Processing using GPUs, pp 48–58
16. Coplin J, Burtscher M (2014) Power characteristics of irregular gpgpu programs. In: Workshop on Green Programming, Computing, and Data Processing (GPCDP), pp 1–6

17. Coplin J, Burtscher M (2016) Energy, power, and performance characterization of GPGPU benchmark programs. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, pp 1190–1199
18. Dally B (2011) Power, programmability, and granularity: The challenges of exascale computing. In: 2011 IEEE International Test Conference, IEEE Computer Society, pp 12–12
19. Dlamini G, Jolha F, Kholmatova Z, Succi G (2022) Meta-analytical comparison of energy consumed by two sorting algorithms. Inf Sci 582:767–777
20. Ferro M, Yokoyama A, Klh V, Silva G, Gandra R, Bragana R, Bulcao A, Schulze B (2017) Analysis of gpu power consumption using internal sensors. In: Proceedings of the 16th WPerformance - Workshop em De- sempenho de Sistemas Computacionais e de Comunicao
21. Ikram MJ, Abulnaja OA, Saleh ME, Al-Hashimi MA (2017) Measuring power and energy consumption of programs running on kepler gpus. In: 2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT), IEEE, pp 18–25
22. Jamal F, Aljabri N, Al-Hashimi M, Saleh M, Abulnaja O (2022) Towards power efficient algorithms: matrix multiplication in high performance computing. Comput Electric Eng (accepted for publication)
23. Kasichayanula K, Terpstra D, Luszczek P, Tomov S, Moore S, Peterson GD (2012) Power aware computing on gpus. In: 2012 Symposium on Application Accelerators in High-Performance Computing, IEEE, pp 64–73
24. Kondo M, Miyoshi I, Inoue K, Miwa S (2019) Power management framework for post-petascale supercomputers. In: Sato M (ed) Advanced Software Technologies for Post-Peta Scale Computing. Springer, pp 249–269
25. Lee S, Kim K, Koo G, Jeon H, Ro WW, Annavaram M (2015) Warped- compression: enabling power efficient gpus through register compression. ACM SIGARCH Comput Architect News 43(3S):502–514
26. Lim J, Lakshminarayana NB, Kim H, Song W, Yalamanchili S, Sung W (2014) Power modeling for gpu architectures using mcpat. ACM Trans Des Autom Electron Syst (TODAES) 19(3):1–24
27. Luo C, Suda R (2011) A performance and energy consumption analytical model for gpu. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, IEEE, pp 658–665
28. Montanana Aliaga JM, Cheptsov A, Hervás A (2021) Towards energy efficient computing based on the estimation of energy consumption. Sustained Simulation Performance 2019 and 2020. Springer, pp 21–33
29. Nagasaka H, Maruyama N, Nukada A, Endo T, Matsuoka S (2010) Statistical power modeling of gpu kernels using performance counters. In: International conference on green computing, IEEE, pp 115–122
30. NVIDIA (date accessed October 10, 2018) NVIDIA Kepler Architecture. URL https://www.nvidia.com/content/PDF/kepler/ Tesla-K40-Active-Board-Spec-BD-06949–001_v03.pdf
31. NVIDIA (date accessed April 11, 2019) NVIDIA Management Library.URL https://developer.nvidia.com/ nvidia-management-library-nvml
32. NVIDIA (date accessed April 11, 2019) NVIDIA System Management Interface. URL https://developer.nvidia.com/ nvidia-system-management-interface
33. NVIDIA (date accessed April 20, 2020) NVIDIA CUDA Programming Guide. URL http://docs.nvidia.com/cuda/ cuda-c-programming-guide/
34. NVIDIA (date accessed January 13, 2019) Advanced Quick- sort. URL http://docs.nvidia.com/cuda/cuda-samples/index.html#advanced-quicksort-cuda-dynamic-parallelism
35. NVIDIA (date accessed January 23, 2019) Vector Addition. URL http://docs.nvidia.com/cuda/cuda-samples/index.html# vector-addition-driver-api
36. NVIDIA (date accessed March 21, 2019) Nsight Visual Studio Edition. URL http://docs.nvidia.com/nsight-visual-studio-edition/5.4/ Nsight_Visual_Studio_Edition_User_Guide.htm#Analysis_Tools_ Overview.htm%3FTocPath%3DAnalysis%2520Tools%7C
37. NVIDIA (date accessed November 20, 2017) GPU Applications. URL http://www.nvidia.com/object/gpu-applications.html
38. Obrien K, Pietri I, Reddy R, Lastovetsky A, Sakellariou R (2017) A survey of power and energy predictive models in hpc systems and applications. ACM Comput Surv (CSUR) 50(3):1–38
39. Origin (date accessed June 11, 2019) Origin Lab. URL https://www.originlab.com/

40. Padoin EL, Pilla LL, Boito FZ, Kassick RV, Velho P, Navaux PO (2013) Evaluating application performance and energy consumption on hybrid cpu+gpu architecture. Clust Comput 16(3):511–525
41. Padoin EL, Pilla LL, Castro M, Navaux PO, M´ehaut JF (2016) Exploration of load balancing thresholds to save energy on iterative applications. In: Latin American High-Performance Computing Conference, Springer, pp 76–88
42. Pool J, Lastra A, Singh M (2010) An energy model for graphics processing units. In: 2010 IEEE International Conference on Computer Design, IEEE, pp 409–416
43. Reed DA, Dongarra J (2015) Exascale computing and big data. Commun ACM 58(7):56–68
44. Roy S, Rudra A, Verma A (2013) An energy complexity model for algorithms. In: Proceedings of the 4th conference on Innovations in Theoretical Computer Science, pp 283–304
45. Roy S, Rudra A, Verma A (2014) Energy aware algorithmic engineering. In: 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, IEEE, pp 321– 330
46. Springmeyer R, Still C, Schulz M, Ahrens J, Hemmert S, Minnich R, Mc- Cormick P, Ward L, Knoll D (2011) From petascale to exascale: eight focus areas of r & d challenges for hpc simulation environments. Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States)
47. Suda R, et al. (2009) Accurate measurements and precise modeling of power dissipation of cuda kernels toward power optimized high-performance cpu-gpu computing. In: 2009 international conference on parallel and distributed computing, applications and technologies, IEEE, pp 432– 438
48. Suda R, et al. (2010) Investigation on the power efficiency of multi-core and gpu processing element in large scale simd computation with cuda. In: International Conference on Green Computing, IEEE, pp 309–316
49. Top500 (date accessed April 20, 2020) Top500 Supercomputers. URL https://www.top500.org/lists/2019/11/
50. Top500 (date accessed April 20, 2020) Top500 Supercomputers. URL https://www.top500.org/green500/lists/2019/11/
51. Ukidave Y, Ziabari AK, Mistry P, Schirner G, Kaeli D (2014) Analyzing power efficiency of optimization techniques and algorithm design methods for applications on heterogeneous platforms. The Int J High-Perform Comput Appl 28(3):319–334
52. Villa O, Johnson DR, Oconnor M, Bolotin E, Nellans D, Luitjens J, Sakharnykh N, Wang P, Micikevicius P, Scudiero A, et al. (2014) Scaling the power wall: a path to exascale. In: SC'14: Proceedings of the International Conference for High-Performance Computing, Networking, Storage and Analysis, IEEE, pp 830–841
53. Ye W, Vijaykrishnan N, Kandemir M, Irwin MJ (2000) The design and use of simplepower: a cycle-accurate energy estimation tool. In: Proceedings of the 37th Annual Design Automation Conference, pp 340–345
54. Zecena I, Zong Z, Ge R, Jin T, Chen Z, Qiu M (2012) Energy consumption analysis of parallel sorting algorithms running on multicore systems. In: 2012 International Green Computing Conference (IGCC), IEEE, pp 1–6
55. Zhang Y, Hu Y, Li B, Peng L (2011) Performance and power analysis of ati gpu: A statistical approach. In: 2011 IEEE Sixth International Conference on Networking, Architecture, and Storage, IEEE, pp 149–158
56. Zhao D, Chen Q (2019) Current prediction model of gpu oriented to general purpose computing. IEEE Access 7:127920–127931