IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 15, 2018, accepted June 26, 2018, date of publication July 31, 2018, date of current version August 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2861571

Analyzing Power and Energy Efficiency of Bitonic Mergesort Based on Performance Evaluation

OSAMA AHMED ABULNAJA, MUHAMMAD JAWAD IKRAM, MUHAMMAD ABDULHAMID AL-HASHIMI, AND MOSTAFA ELSAYED SALEH

Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia

Corresponding author: Muhammad Jawad Ikram (mjawad.kau@gmail.com)

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

ABSTRACT In *graphics processing unit* (*GPU*) computing community, *bitonic mergesort (BM)* is recognized as one of the most investigated sorting algorithms. It is specially designed for parallel architectures, requires minor inter-process communication, can be implemented in-place, and is logically appropriate for *single instructions multiple data* platforms. In addition, GPUs have shown tremendous improvements in power and performance efficiency and thus have become essential ingredients in pursuit of the prospective *exascale* systems whose major obstacle is the excessive power consumption. In a recent research work, we found that fundamental software building blocks can offer a reasonable amount of power and energy saving that can offer new ways to tackle the power obstacle of the prospective exascale systems. We evaluated *average peak power*, *average energy*, and *average kernel runtime* of BM under various workloads and compared it with *advanced quicksort (AQ)*. The results showed that BM outperformed AQ based on all the three metrics in most cases. In this paper, we further investigate BM to identify the factors that result in its underlying power and energy efficiency advantage over AQ. We analyze the power and energy efficiency of BM and AQ based on their performance evaluation on NVIDIA K40 GPU. The performance of both the algorithms is investigated using various experiments offered by *NVIDIA Nsight Visual Studio*.

INDEX TERMS Energy measurement, power measurement, exascale computing, GPU, sorting.

I. INTRODUCTION

One of the main obstacles to reach exascale performance (order 10^{18} floating point operations per second) is the excessive power consumption. Innovations and improvements are required to reach exascale performance in a reasonable power budget. Recently, *graphics processing unit (GPU)* has become a perfect choice for exascale research due to its developments in power and performance efficiency. Scaling the current petascale *high-performance computing* (*HPC*) machines on top500 list [1] to exascale would consume power in Gigawatts, and providing such huge amount of power would need a nuclear power plant of medium size [2]. It is estimated in a report of the US Defense Advanced Research Projects Agency (DARPA) that the maximum *peak power* of upcoming HPC machines must be lower than 20 Megawatts [3]. Huge investment in various areas of research and development is required to address these challenges.

Existing techniques are not appropriate for exascale computing and alternative methods are required that can deal with limitations of energy consumption. It is suggested in our recent research [4] that evaluating power and energy consumption of fundamental software building blocks can provide new techniques to address the power obstacle of the upcoming exascale systems. We found that basic software building blocks can offer a reasonable amount of power and energy saving. In [4], we evaluated power and energy efficiency of *Bitonic Mergesort (BM)* [5] under various workloads and compared it with *Advanced Quicksort (AQ)* [6]. The results compared BM with AQ based on three metrics, as stated in [4]:

- *"Peak Power:* is the peak level of GPU power that is reached when the algorithm (AQ or BM) is executing on the GPU (GPU active state). It is obtained from the power profile of the algorithm.
- *Energy:* is the total energy consumed by the algorithm (AQ or BM). It is indicated by the area under the power curve of the algorithm. It is calculated by integrating the power curve over *kernel runtime*.
- *Kernel Runtime:* is the runtime of a kernel (AQ kernel or BM kernel) that is executing on the GPU. It is the time in which a kernel keeps GPU busy and as a

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

result the GPU consumes more power than its idle state power."

In [4], it is demonstrated that a simple BM outperforms a performance-optimized AQ based on *kernel runtime, peak power,* and *energy*. In this paper, we further evaluate BM and AQ based on various performance metrics and identified that the underlying power and energy consumption advantage of BM occurs due to its efficient execution on the GPU because BM is an example of data-independent sorting algorithm intended to execute on fragment processors whose output location cannot be altered in memory based on input sequence [7]. In contrast, AQ is an example of data-driven algorithms, which are completely dependent on the input sequence. Data-independent sorting algorithms (such as BM) look logically more suitable for GPUs and other parallel platforms because the points of communication are known in advanced. Furthermore, BM is specially designed for parallel architectures, requires minor inter-process communication, can be implemented in-place, and is logically appropriate for the *single instructions multiple data (SIMD)* platforms [7]. In GPU computing community, BM is recognized as one of the most investigated sorting algorithms [4], [7]–[12]. It is also used as construction technique for developing sorting networks that comprise $O(n \log^2(n))$ comparators and having a delay of $O(n \log^2(n))$, where n is the number of elements in the list to be sorted [10]. In contrast, AQ is a parallel quicksort (available in CUDA SDK) based on *CUDA dynamic parallelism* [13]. It is supported on all NVIDIA GPUs having compute capability 3.5 or greater.

Power consumption of a program executing on the GPU is related to its performance on the GPU, as stated in [14], "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." Furthermore, it is demonstrated in [15] that changes in the implementation of GPU code not only significantly result in better performance, but also result in several times improvements in energy consumption and reduce the power consumption by over a factor of two. On the other hand, in [16] CUDA performance counters are used to obtain performance profiles of kernels and linear correlation is observed between the performance profiles and power consumption by statistical model learning. Furthermore, in [17], it is stated that "Power performance of a CUDA processing element (PE) is dependent on electrical features of the inside hardware components and their interconnections; also high-level applications and the parallel algorithms performed on it."

Inspired by the observations in these studies, we further investigate BM based on its performance evaluation to explore its underlying power and energy consumption advantage over AQ. We investigate BM under various experiments offered by *NVIDIA Nsight Visual Studio* [18]. These experiments are used for performance analysis of kernels executing on GPU and determine how kernels use GPU resources and identify performance bottlenecks. We execute four of these experiments both on BM and AQ in order to compare their performance on K40 GPU. The results of each experiment provide a comprehensive comparison of both the algorithms based on a number of performance metrics. Then, we provide a general background on power consumption of GPUs and finally analyze power and energy consumption of BM and AQ based on their performance on the K40 GPU. We tested BM and AQ on 7 different datasets of unsigned integer random numbers. The 7 datasets include 2Mega (M) elements, 16M elements, 32M elements, 64M elements, 128M elements, 256M elements, and 512M elements as described in Table 1. The datasets were generated in the same way as in [4].

TABLE 1. Datasets of unsigned integer random numbers [4].

Datasets	Number of Elements
1	2M = 2^{21} = 2,097,152
2	16M = 2^{24} = 16,777,216
3	32M = 2^{25} = 33,554,432
4	64M = 2^{26} = 67,108,864
5	128M = 2^{27} = 134,217,728
6	256M = 2^{28} = 268,435,456
7	512M = 2^{29} = 536,870,912

We carried out the experiments on a Fujitsu HPC workstation with a dedicated NVIDIA Tesla K40c GPU. NVIDIA Kepler (Tesla K40c) [19] can be found in various top500 [1] and green500 [20] supercomputers. Tesla K40 contains GK110 chip that is composed of 7.1 billion transistors. GK110 is specifically designed to provide outstanding with power and performance efficiency in order to tackle the most devastating challenges in high-performance computing. GK110 chip is capable to accomplish integer, single precision, and double precision performance and high memory bandwidth [19]. Due to the innovative computing technology, Tesla K40 GPU is used by a number of scientific applications [20]. Streaming Multiprocessors (SM), dynamic parallelism [13], and hyper-Q [19] are three unique innovations in Kepler Tesla K40 architecture [19]. Other popular Kepler architecture includes Tesla K20 series, and Tesla K80 series GPUs. Additionally, Microsoft Visual Studio 2013, and NVIDIA Nsight Visual Studio Edition 5.2 were used to successfully accomplish the experiments. Table 2 shows the specifications of the Fujitsu workstation that we used in the experiments. We repeated some of our experiments on another K40c GPU in order to make sure that we obtain the identical results, which we obtained. We used CUDA programming standard in order to benefit from the parallel programming capabilities offered by NVIDIA GPUs. Below is a brief description of the relevant CUDA terminologies [23]:

TABLE 2. System specifications [4].

System Manufacturer	FUJITSU
System Model	CELSIUS M720 Power
Processor	Intel(R) Xeon(R) CPU E5-2640 2.50GHz, 2494 MHz,
Operating System	Microsoft Windows 8.1 x64-based
Physical Memory	8.00 GB

- *Host:* is the CPU and its memory. In case of our experiments, the host is Intel(R) Xeon(R) CPU E5-2640 2.50GHz and its memory.
- *Device:* means the GPU and its memory. In the context of our experiments, the device is the NVIDIA Kepler Tesla K40c GPU and its memory.
- *Kernel:* means a function that is executed on the device. Programs are executed in parallel on the device (GPU) as kernels. A kernel is executed at a time by various threads. In case of our experiments, we have AQ and BM kernels that contain the underlying source codes of the algorithms.
- *Warp:* means a group of 32 threads that are consecutively numbered within a thread block.
- *Barrier:* in the source code, the barrier for a group of threads or process is a point at which threads or process must stop execution and cannot proceed until all other threads or processes are reached at this point.

In general, the contributions of this paper are as follow:

- This paper provides detailed experimental investigation of *power, energy* and *performance* efficiency of BM on K40 GPU, compares it with AQ, and highlights some performance factors that result in high power and energy consumption.
- This paper justifies the *power* and *energy* consumption advantage of BM over AQ based on the experimental findings, and *power* and *performance* analysis.

The rest of this paper is organized as follows. Section II presents adequate literature review of related research work. Section III presents background on performance and power consumption on GPUs. Section IV provides a discussion on results evaluation. Finally, Section V concludes the whole work and provides some future research directions.

II. LITERATURE REVIEW

Al-Hashimi *et al.* [4] provided an experimental investigation of Bitonic Mergesort (BM) for power and energy consumption and compared it with Advanced Quicksort (AQ) on NVIDIA Tesla K40 GPU. The researchers identified that a simple BM provides a reasonable power and energy advantage over an optimized quicksort algorithm in most cases. The study identified that other fundamental software building blocks could also be investigated for power and energy advantage in order to provide better recommendations for the upcoming exascale systems.

Ikram *et al.* [8] proposed a methodology for measuring power and energy consumption of programs executing on NVIDIA Kepler GPUs. They used Kepler K40 GPU as a test platform and studied power and energy consumption of a Bitonic Mergesort program and a Matrix Multiplication program under various workloads.

Al-Hashimi *et al.* [24] investigated the effect of three control loops (*for, while* and *do-while*) on system power consumption and identified *for loop* the most power efficient. Al-Hashimi *et al.* [25] evaluated power and energy consumption of a generic mergesort and an optimized quicksort on Intel Xeon E5-2640 (Sandy Bridge) CPU. The researchers identified some power advantage in mergesort over quicksort.

O'brien *et al.* [26] presented a survey of power and energy predictive models in HPC systems and applications. The researchers highlighted the weaknesses of the power and energy efficiency models to precisely predict the power and energy consumptions by considering the hierarchical and heterogeneous behavior of tightly integrated HPC systems. Bridges *et al.* [27] presented a survey of GPU power modeling and profiling methods with focusing more on significant efforts in this area. They discussed GPU internal and external sensors for evaluation of GPU power consumption. The researchers also discussed developments and challenges of counter-based GPU power modeling.

Padoin *et al.* [28] explored load balancing thresholds for saving energy on iterative applications. The researchers proposed two variants of a novel energy-aware load balancer that aim to reduce the energy consumption of parallel architectures running imbalanced scientific applications without performance degradation. The researchers identified the trade-off between runtime, power demand and total energy consumption while applying two energy-aware load balancer variants on real-world applications.

Vijaykrishnan *et al.* [29] and Kasichayanula *et al.* [30], analyzed power consumption of applications on graphics processing units and other heterogeneous platforms. Nagasaka *et al.* [16] proposed statistical models for GPUs based on vendor provided performance counters. Kasichayanula *et al.* [30] provided an investigation on numerous microbenchmarks executing on GPUs from power consumption perspective.

Zecena *et al.* [31] studied performance and energy consumption both serial and parallel versions of odd-even sort, shell sort, and quicksort on a shared memory system containing two quad-core AMD 2380 Opteron processors. They used iterative versions of odd-even sort and shell sort while for quicksort, they used recursive implementation.

Ukidave *et al.* [32] investigated a number of optimization techniques for power and performance efficiency on various heterogeneous platforms including discrete GPUs, shared memory GPUs, low power system-on-chip (SoC) devices and includes hardware from NVIDIA, Intel and Qualcomm. They measured the energy consumption of optimization techniques

in order to evaluate the tradeoff between power and performance. The researchers identified that architectural and algorithmic factors have an effect on power consumption. They showed that algorithms that implement the same operation can exhibit different performance based on target hardware and design of an application.

Burtscher *et al.* [33] proposed a methodology for power and energy measurement on NVIDIA Kepler K20 GPU using the on-board sensor. The researchers identified a number of anomalies in power and energy measurements. They used multiple systems, GPUs, CUDA programs, and scenarios to validate their methodology. Burtscher and Coplin [34] studied irregular and regular programs executing on K20 GPU from power and energy consumption perspective. They examined the power profile with varying GPU's core and memory frequencies, using different versions of the same algorithm, and varying the program's input. The researchers identified that power behavior of irregular programs cannot be measured accurately with a single average but it must be considered as a function of time and should be evaluated each input after changing the code.

Padoin *et al.* [35] investigated performance and energy consumption of application on hybrid CPU and GPU architecture. The researchers used a scientific application from the agroforestry domain as a case-study and identified how the workload of the application may affect energy efficiency on hybrid architectures.

Roy *et al.* [36] suggested energy measurement as a basis for designing and implementing algorithms. The researchers found memory parallelism as a factor that has an effect on energy consumption of algorithm. For validation, they used asymptotic energy complexity model [37]. The researchers found that the energy optimal layout (8-way parallel) of selection sort have better performance than non-optimal (1-way parallel). They also found a reasonable energy saving in mergesort and quicksort can be achieved due to changes in parallelization.

Vila *et al.* [38] demonstrated the research conducted be NVIDIA Corporation towards designing exascale systems by integrating features that address the scaling problems of performance and energy efficiency. They concluded that more innovations in algorithms and architecture will be required for improvements in the performance of applications in order to improve memory locality, better scaling, and integer execution efficiency.

Dally [39] explained the challenges for the upcoming exascale systems and current methods to address those challenges. He demonstrated that 200-fold improvement will be required in energy per instruction in order to obtain exascale performance in 20 Megawatts power budget. He further explained that more creative programming environment will be required for programming exascale systems.

Suda and Ren [40] proposed two models for predicting execution time and energy consumption. Their models enable programmers to have a better understanding of the performance and energy-saving bottleneck of parallel applications on GPU architectures. Ren and Suda [17] proposed a method for load sharing in order to adjust the workload assignment within the CPU and GPU components inside a CUDA *processing element (PE)* with the objective of optimizing the overall power efficiency.

Lim *et al.* [41] developed a power model based on McPAT [42] for GPUs. In their model, initial data was generated from McPAT that contained GPU configuration in detail. Afterwards, the model was adjusted by comparing them with empirical data. NVIDIA's Fermi architecture was used for developing the power model. Chen *et al.* [42] proposed an integrated power and performance (IPP) prediction model for GPU architecture. For a given application, their model can predict the optimal number of active processors. The researchers argued that improvement in application performance cannot be achieved when an application reaches the peak memory bandwidth and using more cores. Their model can also predict the runtime events occurring on the GPU, unlike previous models.

Zhang *et al.* [43] used a tree-based random forest technique for developing a model for power consumption. Compared to regression-based techniques, their model achieved better accuracy for investigating correlation among individual performance metrics. Similarly, for ATI GPUs, Pool *et al.* [44] also used random forest techniques and investigated performance and power consumption together. On the other hand, Lee *et al.* [45] used a technique in which energy and power model was developed based on the unit energy consumed by each instruction.

Connors and Qasem [46] exploited value similarity for improving power consumption of register file. The researchers proposed a warp compression scheme for GPU register files on the basis of similarity of register values that resulted in a reduction of both dynamic and leakage power.

Coplin and Burtscher [47] studied the impact of thread block size and register allocation on the performance and power consumption of three heuristic search algorithms. They observed significant variations in performance for codes executing at the same level of occupancy. The best power-performance trade-offs were not always achieved at the highest level of occupancy. The study also found that, for a given occupancy level, having larger thread blocks, and consequently fewer thread blocks per SM, generally leads to improvements in both performance and power consumption.

The researchers studied source-code optimizations and their effects GPU performance, power draw, and energy consumption. The researchers investigated 128 versions of two n-body codes and measured the active runtime and the power consumption of each code version on three inputs, various GPU clock frequencies, two arithmetic precisions, and with and without ECC. They showed that performance and energy consumption of a GPU kernel can be altered by a factor of up to five using individual and combinations of optimizations. Coplin and Burtscher [15] studied the energy consumption, power draw, and runtime of 34 programs from 5 general purpose GPU benchmark suites. Ferro *et al.* [48] employed

the GPU sensors to obtain high-resolution power profiles of real and benchmark applications. The researchers wrote their own tools to query the sensors of two NVIDIA GPUs from different generations and compare the accuracy of them. They also compared the power profile of GPU with CPU using IPMItool.

Inspired by these prior studies, this paper takes an orthogonal approach to exploit fundamental software building blocks such as sorting algorithms for the inherent algorithmic power and energy efficiency. In this regard, we previously investigated *kernel runtime* and power and energy efficiency of BM and compared it with AQ [4]. In this paper, we further analyze the power and energy efficiency of BM based on its performance on the GPU to identify its inherent algorithmic power and energy consumption advantage and open insights for exascale research.

III. BACKGROUND ON PERFORMANCE ANALYSIS AND POWER CONSUMPTION ON GPUS

In this section, we provide background information on performance and power consumption of applications executing on GPUs. We provide background on performance based on various experiment offered by *NVIDIA Nsight Visual Studio Edition*. Background on power consumption is discussed based on analytical modeling.

A. BACKGROUND ON PERFORMANCE

NVIDIA Nsight Visual Studio Edition 5.2 [18] is a development environment for CUDA and graphics applications running on NVIDIA GPUs, which is integrated into Microsoft Visual Studio. It provides a number of experiments that can be used to know how the kernels got executed on the device and how they utilized the device resources and capabilities. We performed four experiments on AQ and BM that include *occupancy experiment, issue efficiency experiment, instruction statistics experiment,* and *memory statistics experiment.* These experiments are described below [18]:

Occupancy Experiment: *determines the achieved occupancy during kernel runtime. Achieved occupancy* is determined by obtaining the ratio of *active warps* in each *streaming multiprocessor (SM)* to the maximum possible active warps (on an SM of K40 GPU, maximum active warps = 64). 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 represents the *achieved occupancy* on an SM for a kernel executing on the device:

$$Achieved\ occupancy = \frac{active\ warps}{maximum\ active\ warps} \quad (1)$$

If *achieved occupancy* is low, it leads to poor instruction *issue efficiency* because of less *eligible warps* to hide latency between dependent instructions. On the other hand, when *achieved occupancy* is at an adequate level to hide latency, further increasing it has a negative effect on the performance of a kernel due to reduction of resources per thread [18]. Investigating *achieved occupancy* and observing its effect on *kernel runtime* when running at different occupancy levels should be one of the initial steps in analyzing kernel's performance on the device. The causes of low *achieved occupancy* include unbalanced workload within the blocks and across the blocks and launching fewer blocks. The values reported for *achieved occupancy* in Section IV are the average across all *warp schedulers* [23] for the duration of the kernel execution. An SM contains one or more *warp schedulers*, each of which attempts to issue instructions from a warp on each clock cycle. To sufficiently hide latencies between dependent instructions, each scheduler must have at least one warp eligible to issue an instruction every clock cycle. Keeping occupancy at a high level throughout *kernel runtime* helps to avoid situations where all warps are stalled and no instructions are issued (this is explained in detail in *issue efficiency experiment*). *Achieved occupancy* per SM over the *kernel runtime* is obtained as follows. Firstly, occupancy is measured on each *warp scheduler* based on performance counters to count the number of *active warps* on that scheduler every clock cycle. Secondly, to find the average *active warps* across each SM, these counts are added across all *warp schedulers* on each SM and divided by the clock cycles the SM is active. Finally, *achieved occupancy* is obtained per SM averaged over the *kernel runtime* based on equation 1, i.e. dividing average *active warps per SM* by the SM's maximum supported number of *active warps* [18].

Issue Efficiency Experiment: *provides information about the device's ability to issue the instructions.* This experiment collects results for *warps per SM*, *warp issue efficiency*, and *issue stall reasons* as described below [18].

- Warps per SM: *provides an overview of the various warps per SM metrics.* The metrics are reported as average values across the complete kernel execution for each individual SM of the target device. The device limit for K40 GPU is 64. The following metrics are collected for *warps per SM* in *issue efficiency* experiment.
  - Active Warps: *A warp is active from the time it is scheduled on a multiprocessor until it completes the last instruction.* Each *warp scheduler* maintains its own list of assigned active warps. This assignment of warps to the *schedulers* is done once at the time a warp becomes active and is valid for the lifetime of the warp. There should be at least a minimum of eight *active* warps per *warp scheduler*. More *active warps* might allow hiding warp latencies more efficiently.
  - Eligible Warps: *An active warp is considered eligible if it is able to issue the next instruction*. Each *warp scheduler* will select the next warp to issue an instruction from the pool of *eligible warps*. Warps that are not *eligible* will report an *issue stall reason*.
- Warp Issue Efficiency: *shows the distribution of the availability of eligible warps per cycle across the GPU.* The values are reported as the sum across all *warp schedulers* for the duration of the kernel execution. The

following metrics are collected for *warp issue efficiency* in *issue efficiency* experiment [18].

- No Eligible: *shows the number of cycles that a warp scheduler had no eligible warps to select from and therefore did not issue an instruction*. The lower the percentage of cycles with no *eligible warp*, the more efficient the code runs on the target device. We investigate the *issue stall reason* to understand what keeps warps from becoming *eligible* if this value is high.
- One or More Eligible: *shows the number of cycles that a warp scheduler had at least one eligible warp to select from.*

- Issue Stall Reasons: *capture why an active warp is not eligible*. On devices of compute capability 3.0 and higher, every stalled warp increments its most critical stall reason by one on every cycle. The sum of the stall reasons, hence increment per multiprocessor per cycle, by a value between zero (if all warps are eligible) and the number of active warps (if all warps are stalled). In this experiment, values are collected for the following *issue stall reasons* as described below [18].
  - Instruction Fetch: *The next assembly instruction has not yet been fetched.*
  - Pipeline Busy: *The compute resources required by the instruction are not yet available.*
  - Execution Dependency: *An input required by the instruction is not yet available.* Execution dependency stalls can potentially be reduced by increasing instruction-level parallelism.
  - Memory Dependency: *A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding.* Memory dependency stalls can potentially be reduced by optimizing memory alignment and access patterns.
  - Memory Throttle: *A large number of incomplete memory operations obstruct further forward progress.* These can be decreased by bringing together a number of memory transactions into one.
  - Synchronization: *The warp is blocked at a barrier (_syncthreads( ) call).*

Instruction Statistics Experiment*: provides a first level triage for understanding the overall utilization of the target device when executing the kernel.* It answers the following questions: 1) is the kernel grid able to keep all SMs busy over the *kernel runtime*? 2) is a well-balanced distribution of workloads achieved? and 3) does the achieved instruction throughput come close to the hardware's peak performance? This experiment collects results for *SM activity, warps launched, instructions per warp (IPW)*, and *instructions per clock (IPC)*, as described below [18].

- SM Activity: *shows the percentage how long each SM was active during the execution of the kernel*. If an SM is busy in executing at least one *active warp*, it is considered to be active. An SM can be inactive, even though the kernel has not finished its execution because of imbalances in the workload between the SMs, which can occur due to some factors that include different execution times for the kernel blocks, variations between the number of scheduled blocks per SM, or a combination of the two. The imbalance workload across the SMs simply means that some SMs finished the workload execution and become idle while other SMs are still busy in executing the workload. This behavior of SMs is referred to as *tail effect.*
- Warps Launched: *shows the total number of warps launched per SM for the executed kernel grid*. Large variation in the number of *warps launched* across each SM shows lack of adequate amount of parallelism within the kernel grid, which results in poor usage of all available compute resources of the device.
- Instructions per Warp (IPW): *shows the average executed instructions per warp for each SM*.
- Instructions per Clock (IPC): *shows the achieved instructions throughputs per SM for both issued instructions and executed instructions. Issued IPC* and *executed IPC* show the number of issued and executed instructions per cycle accounting for every iteration of *instruction replays* respectively. At every instruction issue time, each *warp scheduler* selects one warp that is able to make forward progress from its assigned list of warps. For this selected warp, the *warp scheduler* then issues either the next single instruction or the next two instructions. A *warp scheduler* might need to issue an instruction multiple times to actually complete the execution of all 32 threads of a warp. Issuing an instruction multiple times is also referred to as *instruction replay*. Each replay iteration takes away the ability to make forward progress by issuing new instructions on that *warp scheduler*. In addition, the compute resources required to process the instruction are consumed for every *instruction replay* [18].

Memory Statistics Experiment: *focuses on the usage of the memory subsystem during kernel execution* [18]. Each individual experiment covers a particular area of the *memory hierarchy* that includes *buffers, caches, atomics, global, local,* and *shared memory spaces* as defined by the CUDA programming model [23].

B. BACKGROUND ON POWER AND ENERGY CONSUMPTION OF GPUS

Power consumption of electronics devices including GPUs can be classified into two parts, which are *static power*, and *dynamic power* as represented by equation 2 [42].

$$Power = Static\ Power + Dynamic\ Power \tag{2}$$

Static power depends on the circuit technology, chip layout, and the operating temperature. On the other hand, *dynamic power* is based on the switching overhead in transistors and it depends on the runtime events on the device [42].

In simple terms, *static power* means *GPU idle state* power, which means the power consumption when GPU is on but no program is running on it. On the other hand, *dynamic power* means the *GPU active state* power, which is the power consumption when the GPU is busy in executing a workload. In [20], we proposed a methodology for measuring power and energy consumption of programs running on Kepler GPUs. The methodology was applied on two programs, which are BM and Matrix Multiplication (MM) program. We obtained power profile of the program using various tools that include NVIDIA System Management Interface (NVSMI), Excel Worksheets, and Origin Software.

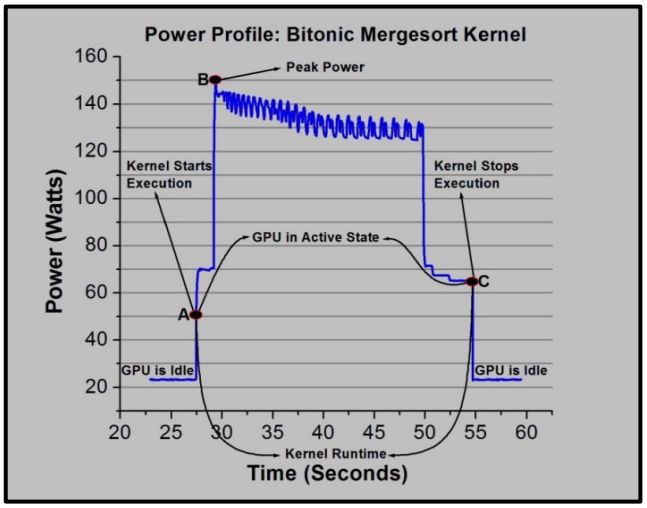


FIGURE 1. GPU full power profile (Idle + Active) for BM: Dataset = 1G elements [20].

Figure 1 shows full power profile of GPU while BM kernel starts and finishes its execution on a dataset having 1Giga (2^{30}) elements [20]. The power profile represents the current *power draw* of the GPU board that is obtained through the built-in GPU sensor at 66.6Hz sampling rate through NVSMI. We have highlighted *GPU idle state, GPU active state*, *kernel runtime,* and *peak power*. Point 'A' in Figure 1 illustrates the point where GPU becomes *active* as workload starts execution on it. As a result, the current *power draw* of the GPU board goes high. We have identified *peak power* by Point 'B', where Point 'C' shows the point when GPU reaches its *idle state* again after executing the kernel. In simple terms, the GPU is in the *active state* only between Point 'A' and Point 'C', and before Point 'A' and after Point 'C', it is in the *idle state*. In *idle state*, the GPU consumes only its *statics power*, which is 20.57W [49] for K40 GPU while in the *active state*, it is consuming *dynamic power*, which depends on many factors including the workload and the algorithm executing on it. The *energy* consumption of the kernel is obtained by integrating the *dynamic power curve* between Point 'A' and Point 'C' as shown in equation 3. The difference between timestamps at Point 'C' and Point 'A' represents the *kernel runtime* as expressed in equations 4. *Peak Power* is represented by Point 'B' in Figure 1, and it is expressed in equation 5.

$$Energy = \int_{t_A}^{t_C} P(t)\,dt, \quad where\ t_A \le t \le t_C \tag{3}$$

$$Kernel\ Runtime = t_C - t_A \tag{4}$$

$$Peak\ Power = max(P(t) = P_{t_B} \tag{5}$$

where $P(t)$ represents the corresponding power curve obtained through GPU built-in sensor.

t_A, t_B, and t_C correspond to timestamps at Point 'A', Point 'B', and Point 'C' respectively.

P_{t_B} shows *peak power*, which represents the maximum value in the power curve and it lies at Point 'B'.

Equation 3 suggests that *energy* consumption of a kernel is directly proportional to its *dynamic power (GPU active state power)* consumption and its *kernel runtime*. In general, *kernel runtime* of a program can be improved by optimizing it for performance but the *dynamic power* consumption is associated with some other factors, which are discussed below.

Chen *et al.* [42] represent the GPU *active state power* as aggregated power across all the components of GPU, which include the *Streaming Multiprocessors (SMs)* and *memory subsystem* of the GPU, as expressed in equation 6.

$$\begin{aligned} Power_{active} &= \sum_{i=0}^{n} Power_{activ\_component_i} \\ &= Power_{active\_SMs} + Power_{active\_Memory} \end{aligned} \tag{6}$$

An SM contains several units, which include *integer arithmetic unit (IAU), floating point unit (FPU), special function unit (SFU), arithmetic logic unit (ALU), texture cache (TC), constant cache (CC), shared memory (SHM), register file (RF)*, and *fetch-decode-schedule (FDS)* unit. The *dynamic power* consumption of a single SM is modeled by equation 7 and the *dynamic power* consumption across all the SMs is given by equation 8 [42].

$$\begin{aligned} &Power_{active\_SM} \\ &= Power_{active\_IAU} + Power_{active\_FPU} + Power_{active\_SFU} \\ &\quad + Power_{active\_ALU} + Power_{active\_TC} + Power_{active\_CC} \\ &\quad + Power_{active\_SHM} + Power_{active\_const\_mem} \\ &\quad + Power_{active\_RF} + Power_{active\_FDS} \end{aligned} \tag{7}$$

$$\begin{aligned} &Power_{active\_all\_SMs} \\ &= N.Power_{active\_SM} \end{aligned} \tag{8}$$

where N is the total number of SMs in a GPU.

On the other hand, the memory subsystem of the GPU contains *global memory*, *shared memory*, *local memory*, *texture memory*, and *constant memory*. *Shared memory, constant memory*, and *texture memory* are modeled as components of SM as they mostly use caches inside an SM as represented in equation 7. The *global memory* and the *local memory* share the same physical *graphics double data rate (GDDR)* memory. The *dynamic power* consumption of the *memory subsystem* is modeled by equation 9 [42].

$$\begin{aligned} &Power_{active\_Memory} \\ &= Power_{active\_global\_mem} + Power_{active\_local\_mem} \end{aligned} \tag{9}$$

TABLE 3. Average peak power, average energy, and average kernel runtime for AQ and BM.

Measurement	Datasets									
	2M		64M		128M		256M		512M	
	AQ	BM	AQ	BM	AQ	BM	AQ	BM	AQ	BM
Average Peak Power (Watts)	73.61	68.64	143.8	137.75	145.04	141.41	145.93	143.73	148.1	144.71
Average Energy (Joules)	156.52	312.03	449.5	373.82	693.19	602.68	1211.02	976.68	2311.13	1670.28
Average Kernel Runtime (Seconds)	2.39	4.82	5.98	4.62	8.75	6.7	14.61	9.88	26.11	15

TABLE 4. Performance results for AQ and BM for all datasets.

Experiments	Performance Metrics	Datasets													
		2M		16M		32M		64M		128M		256M		512M	
		AQ	BM	AQ	BM	AQ	BM	AQ	BM	AQ	BM	AQ	BM	AQ	BM
Occupancy	Achieved Occupancy	63.94%	82.14%	83.10%	82.31%	92.27%	82.38%	93.38%	82.40%	38.57%	82.37%	13.16%	82.39%	NR	82.36%
Issue Efficiency	Warps/SM: Active	44.18	52.62	58.16	52.91	60.72	52.75	25.99	52.73	8.48	52.73	5.22	52.71	1.62	52.71
	Warps/SM: Eligible	4.69	3.37	5.01	3.39	4.94	3.41	0.27	3.41	0.51	3.41	0.92	3.41	0	3.36
	Warp Issue Efficiency: No Eligible	57.63%	65.02%	53.09%	64.59%	51.48%	64.52%	97.21%	64.51%	94.74%	64.46%	90.36%	64.46%	82.69%	64.94%
	Warp Issue Efficiency: One or More Eligible	42.37%	34.98%	46.91%	35.41%	48.52%	35.48%	2.79%	35.49%	5.26%	35.54%	9.64%	35.54%	17.31%	35.06%
Instruction Statistics	SM Activity	97.98%	97.43%	87.04%	99.67%	93.88%	99.83%	96.21%	99.91%	90.86%	99.96%	95.53%	99.98%	NR	99.99%
	Instruction/Warp	258.81	31	347.96	31	353.96	31	385.36	31	411.47	31	432.99	31	569.28	31
	Instruction/Clock: Issued	1.65	1.57	2.12	1.59	2.2	1.6	2.2	1.6	2.2	1.6	2.17	1.6	NR	1.58
	Instruction/Clock: Executed	1.41	1.33	1.8	1.36	1.8	1.36	1.85	1.36	1.85	1.36	1.83	1.36	NR	1.34
	Warps Launched	71245.27	4369.07	600308.8	34952.53	1195480	69905.07	2365237	139810.1	4708073	279620.3	9369296	559240.5	18792389	1118481

Isci and Martonosi [50] presented an empirical method to develop a power model. They studied the runtime power *(dynamic power)* consumption of Intel Pentium 4 processor. Their basic model contains both *static power* and *dynamic power*. The researchers demonstrated that *dynamic power* consumption of an architectural unit has a linear relation with the *access rate* of that component. The *access rate* shows how often the component is accessed per unit of time. In [42], *access rate* for a component of GPU is modeled, which is based on the *dynamic number of instructions per component (DPC)*, *warps per SM*, and *execution cycles* as given in equation 10. The *execution cycles* are divided by 4 because a single instruction is fetched, scheduled, and executed every four cycles.

$$Access\ Rate = \frac{DPC \times Warps\_per\_SM}{execution\ cycles/4} \tag{10}$$

DPC and *Warps per SM* are obtained using equation 11 and 12 respectively [42].

$$DPC = \sum_{i=0}^{n} Number\_of\_Instructions\_per\_Warp_i \times (Component) \tag{11}$$

$$Warp\_per\_SM = \frac{\#threadsPerBlock}{\#threadsPerWarp} \times \frac{\#Blocks}{active\_SMs} \tag{12}$$

IV. RESULTS EVALUATION

In this section, we investigate the power and energy efficiency of Bitonic Mergesort (BM) and Advanced Quicksort (AQ) on NVIDIA Tesla K40 GPU based on the four experiments described in Section III. Table 3 shows results of BM and AQ for 5 datasets (2M, 64M, 128M, 256M, and 512M elements), which are taken from our recently published research [4]. The results provide a comprehensive comparison of BM and AQ based on three metrics, which are *average peak power*, *average energy* and *average kernel runtime* for the 5 datasets. More importantly, for power consumption analysis, we compared the algorithms based on its *peak power* because it is the relevant energy concern for parallel computers as identified by Poon and Sout [51]. It is obvious from results in Table 4 that AQ not only has higher *peak power* and *energy* than BM but also has higher *kernel runtime* in most cases. Higher *kernel runtime* of AQ means that it keeps the GPU busy for a longer time than BM while executing similar workloads. The results reveal that *energy* consumption of AQ and BM is directly associated with *kernel runtime* and the area under the *power curve* of the dataset. BM seems to provide increasingly better energy efficiency as the working set of the workload increases. More importantly, results illustrate that in most cases, BM has a very clear advantage over AQ in terms of *peak power*, *energy* and *kernel runtime*. For instance, for a dataset of 512M elements, *average energy* consumption of AQ and BM is 2311.13J and 1670.28J respectively. This means that on average, BM has 640.85J lower energy consumption than AQ while sorting a dataset of 512M elements.

Figures 2 to 4 show results of *occupancy experiment* for AQ and BM for three datasets respectively. The values reported are the average across all *warp schedulers* for the duration of the kernel execution. In this experiment, we obtained results for all the seven datasets for *achieved occupancy*. For brevity, we only show results of this metric for 3 datasets (64M, 128M, and 256M) for AQ and BM in Figures 2 and 3 respectively. In Figure 4, we show results of *average achieved occupancy* for all the seven datasets. Since

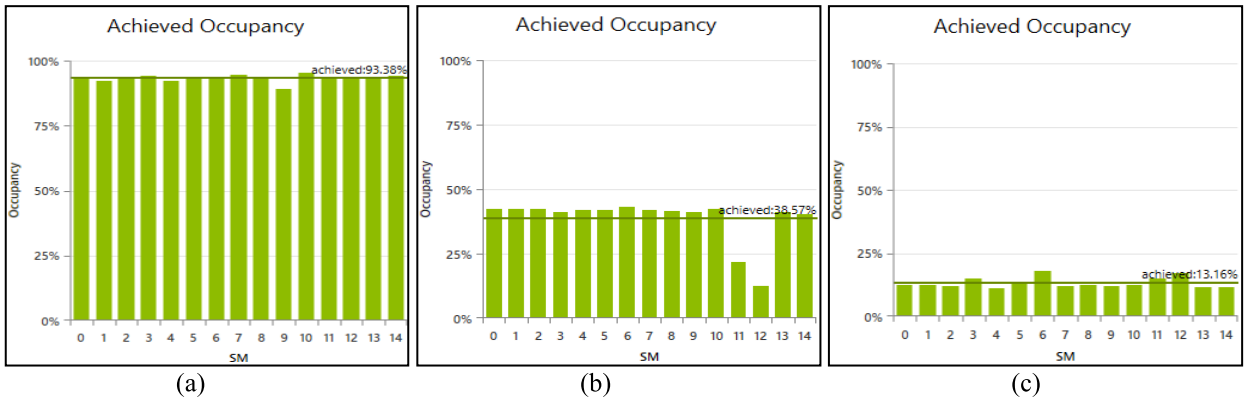


FIGURE 2. Achieved occupancy per SM for AQ: (a) 64M, (b) 128M, (c) 256M.

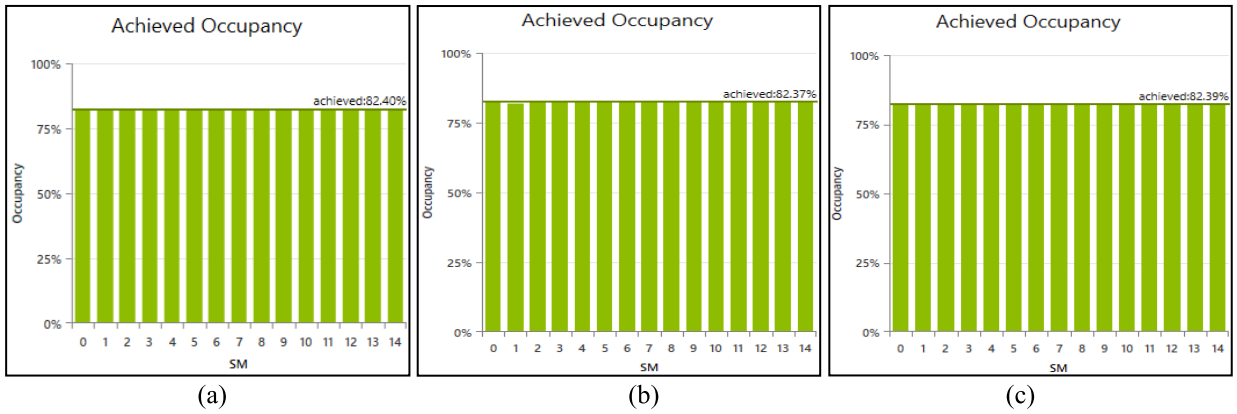


FIGURE 3. Achieved occupancy per SM for BM: (a) 64M, (b) 128M, (c) 256M.

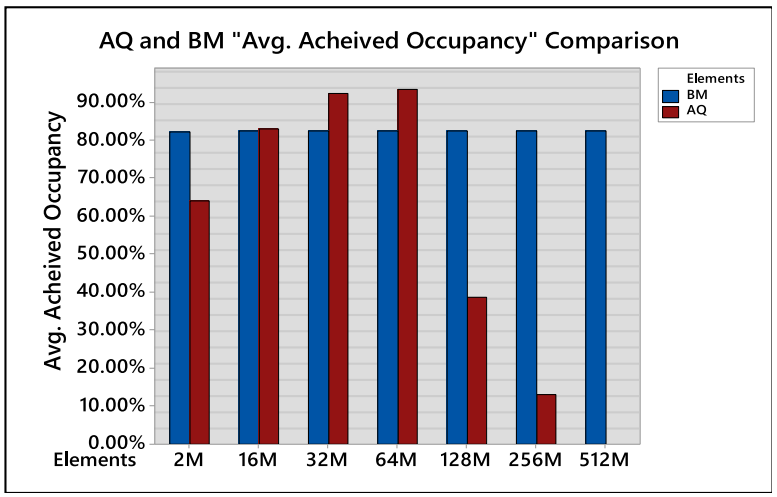


FIGURE 4. AQ and BM average achieved occupancy.

NVIDIA Tesla K40 has 15 *streaming multiprocessors (SMs)*, thus the x-axes in the Figures 2 and 3 show *achieved occupancy* on each SM (SM0 to SM14) of the Tesla K40 GPU. The horizontal line across the bars represents the *average achieved occupancy*. Figures 2 and 3 suggest that BM has uniform *achieved occupancy* across each SM of the GPU while AQ has varying *achieved occupancy* across each SM, particularly for datasets greater than 64M elements. In case of smaller datasets, AQ has also uniform *achieved occupancy* across all the SMs of the GPU like BM, while for datasets greater than 256M elements, the *NVIDIA Nsight Visual Studio* could not collect results of *achieved occupancy* for AQ due

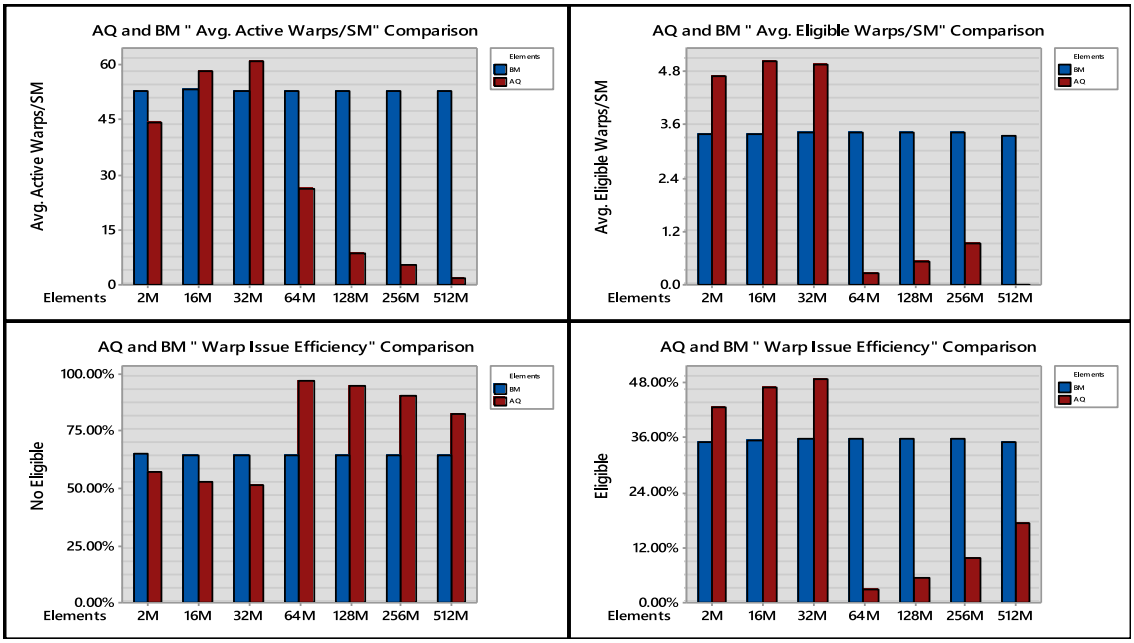


FIGURE 5. AQ and BM issue efficiency: Avg. active warps per SM, Avg. eligible warps per SM, warp issue efficiency (No Eligible), and warp issue efficiency (Eligible).

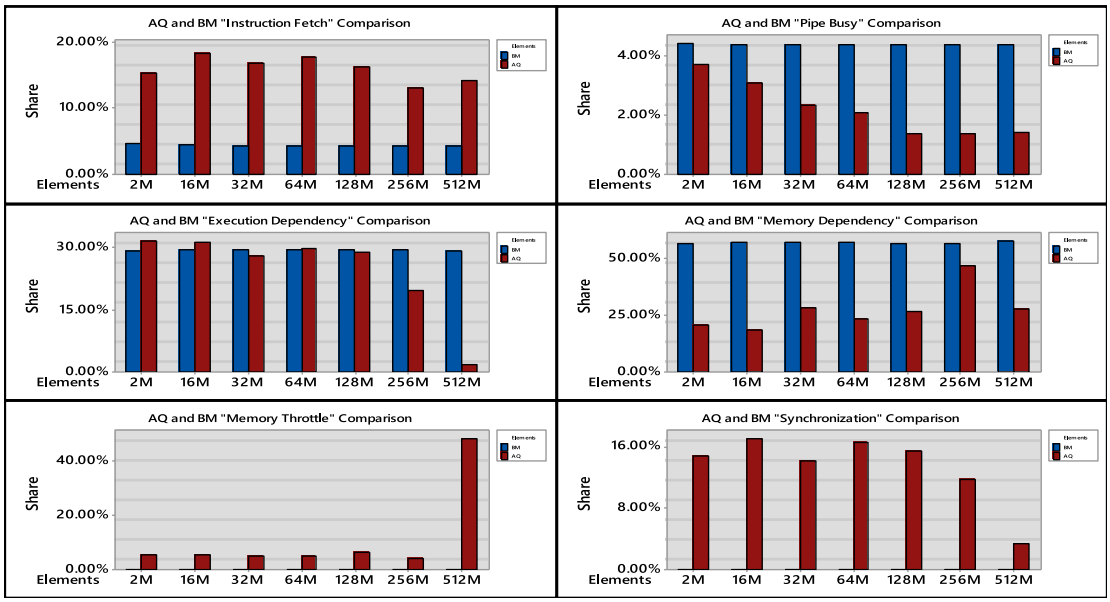


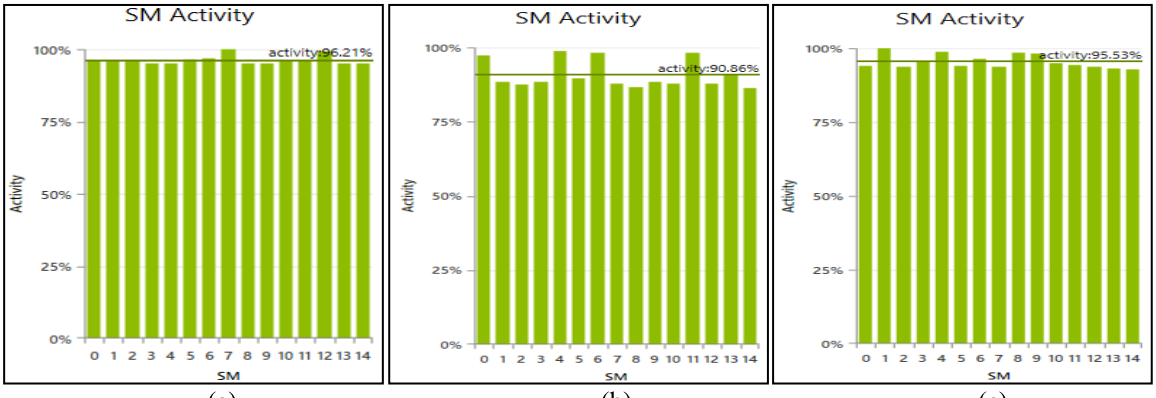
FIGURE 6. AQ and BM issue stall reasons: instruction fetch, pipe busy, execution dependency, memory.

to overflow. The overflow in profiling AQ on larger datasets (datasets $>$ 256M elements) is because all the compute resources of the GPU are oversubscribed as the profiler runs each experiment multiple times to collect results for *achieved occupancy* metric. On the other hand, for BM, overflow in profiling does not occur even for larger datasets due to well-balanced and efficient workload execution on the GPU. Lower and non-uniform *achieved occupancy* of AQ across each SM of the GPU means that there are lower *active warps* across each SM throughout the *kernel runtime*, which results in higher latencies and stalled warps (limit the *warp scheduler* ability to issue instructions from a warp on every clock cycle). Each time a *warp scheduler* is not able to issue an instruction, an *issue stall reason* is recorded. This is explained in detail in *issue efficiency* experiment as shown in Figures 5 and 6. Next, we discuss the impact of *achieved occupancy* on kernel's power and energy consumption. In *occupancy experiment,* the results reveal that BM has comparatively very high *achieved*

occupancy (around 82% on average) while AQ has very low and non-uniform *achieved occupancy* (particularly for larger datasets) across all the SMs throughout the *kernel runtime*. Coplin and Burtscher [47] identified that occupancy has a relation with both performance and power consumption of the code executing on the GPU. They identified that for a certain occupancy level, having larger *thread blocks*, and subsequently smaller *thread blocks per SM*, generally result in improvements in both performance and power consumption. Considering the results of *occupancy experiment*, lower *achieved occupancy* of AQ results in higher latencies and stalled warps, which limit the *warp scheduler* ability to issue instructions from a warp on every clock cycle. Not issuing an instruction from a warp leads to higher *kernel runtime*, which directly affects the *energy* consumption of the algorithm as suggested by equation 3. In addition, due to higher latencies over each SM and prolonged *kernel runtime* of AQ, operating temperature of the chip goes higher that results in an increase in the *static power* of the device. Equation 2 suggests that increase in the *static power* directly affects the total *power* consumption of the device. Thus, the lower and unbalanced *achieved occupancy* of AQ across all the SMs contribute to its higher *peak power, energy,* and *kernel runtime* as compared to BM.

Figure 5 shows a comparison of AQ and BM based on *issue efficiency* experiment. The results suggest that BM executes all the workloads more efficiently than AQ as it has on average more *active* and *eligible* warps across each SM in most cases, particularly in case of larger datasets, BM has a significantly higher number of *active* and *eligible* warps across each SM. AQ has very low *active warps per SM,* which is one of the reasons of having lower *achieved occupancy* as shown in results of *occupancy experiment* (Figures 2 to 4). Equation 1 suggests that the lower the number of *active warps per SM,* the lower will be the *achieved occupancy,* which usually leads to poor performance of kernel on the device. Higher *active* and *eligible warps* across each SM in case of BM hides warp latencies more efficiently. Due to more *eligible warps* across all the SMs in case of BM, the next instructions are issued more efficiently. Since AQ has comparatively fewer *eligible warps* than BM, thus, more *stall reasons* are reported as depicted in *warp issue efficiency* results. Figure 5 shows that for datasets greater than 32M elements, AQ has a very high percentage of *no eligible* warps than BM. The lower the percentage of cycles with *no eligible* warps, the more efficient the code runs on the GPU. In most cases, BM has comparatively lower percentage of *no eligible warps* than AQ while executing the same workload, thus it runs more efficiently on the device having more *eligible warps* throughout the *kernel runtime*.

Figure 6 shows the *issue stall reasons*, i.e. the reasons why an *active warp* is not *eligible*. The *issue efficiency* experiment reports results for six *issue stall reasons* as described in Section III. In case of AQ, the prominent *issue stall reasons* are *execution dependency, memory dependency,* and *instruction fetch.* Two *issue stall reasons,* i.e. *synchronization* and *memory throttle,* are only reported for AQ. On the other hand, the prominent *issue stall reasons* in case of BM are also *memory dependency* and *execution dependency* but other *issue stall reasons* are negligible. Figure 6 shows that for dataset 512M elements, *memory throttle* has 48% contribution in stalling instructions in case of AQ, which occur due to a large number of incomplete memory operations that obstruct further forward progress. Figure 6 also shows that stalls due to *synchronization* only occurs in case of AQ, which happens when a warp is blocked at a *barrier* (*_syncthreads( )*call). The impact of *issue efficiency* on power and energy consumption of the kernel is discussed as follows. The results of *issue efficiency experiment* have a direct relation with *occupancy experiment,* for instance, *active warps per SM* metric is obtained in *issue efficiency experiment* that is used in the calculation of *achieved experiment* as expressed in equation 1. Furthermore, the results of *issue efficiency experiment* also suggest that BM has better execution of the workload on the device as it has comparatively more *eligible warps* across all the SMs than AQ, that efficiently hide warp latencies. This means that unlike AQ, BM finishes the workload execution in lower *kernel runtime* that leads to a lower increase in the



(a) (b) (c)

FIGURE 7. SM Activity for AQ: (a) 64M, (b) 128M, and (c) 256M.

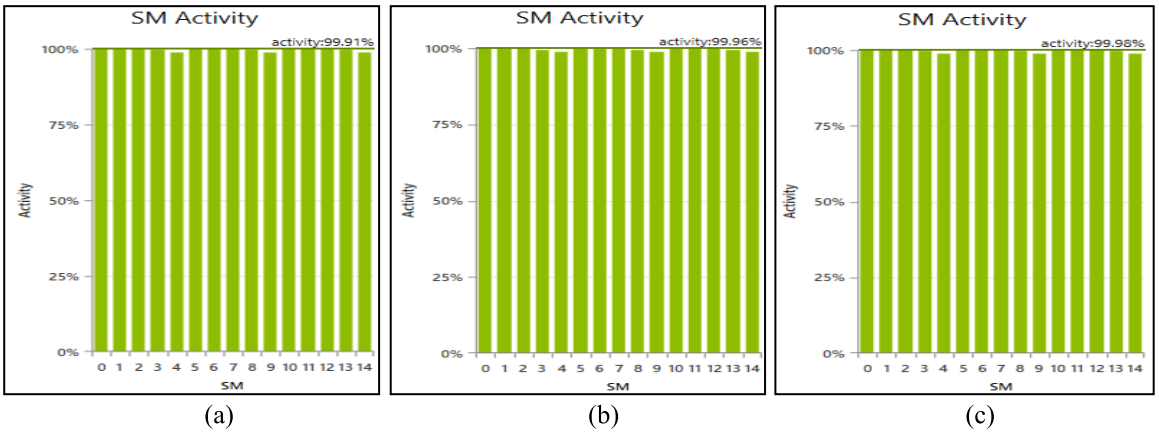


FIGURE 8. SM Activity for BM: (a) 64M, (b) 128M, and (c) 256M.

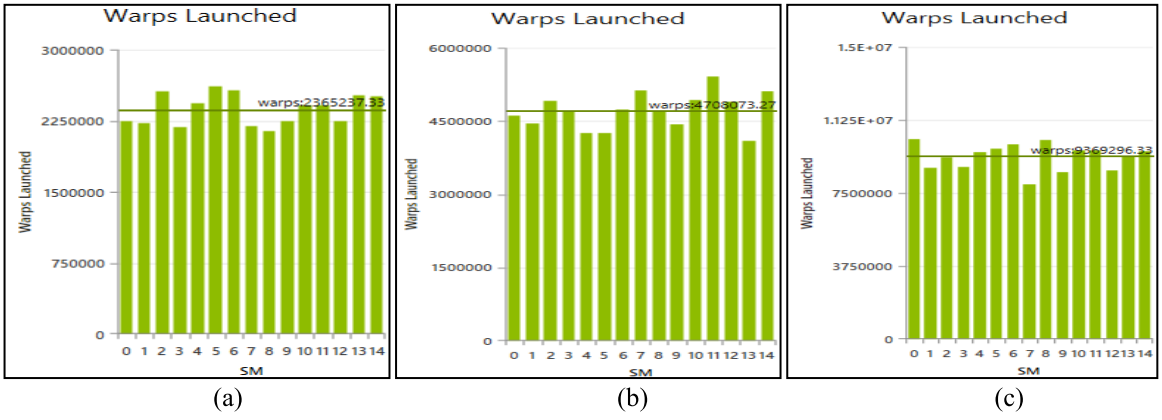


FIGURE 9. Warps launched for AQ: (a) 64M, (b) 128M, and (c) 256M.

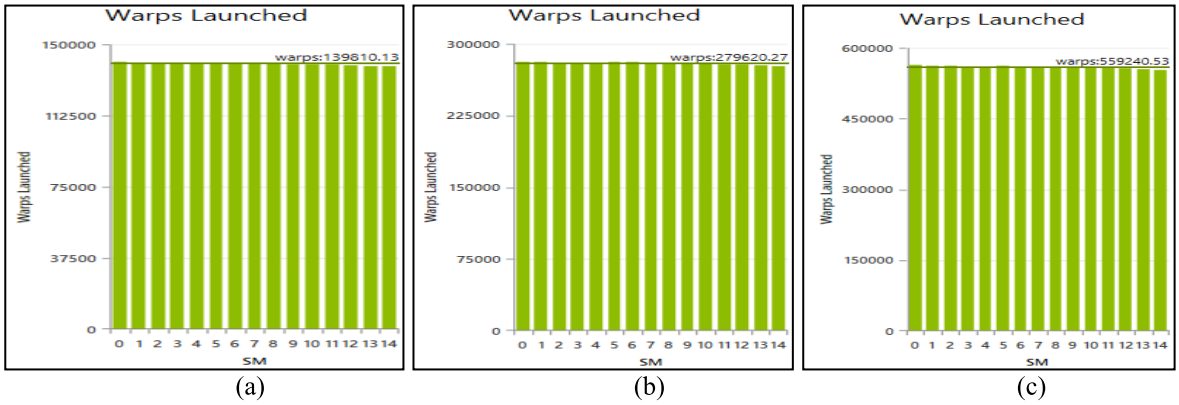


FIGURE 10. Warps launched for BM: (a) 64M, (b) 128M, and (c) 256M.

operating temperature. As a result, there is a slight increase in the *static power* consumption of the device, thus having overall better power and energy efficiency. The increase in *dynamic power* consumption of the GPU due to BM and AQ execution is explained in *instructions statistics* and *memory statistics experiments.*

In Figures 7 to 13, we show results of *instruction statistics experiment* in which we obtained results for all the seven datasets for *SM activity, instruction per clock, warps per instruction, and warps launched*. For brevity, we only show results of these metrics for 3 datasets (64M, 128M, and 256M) in Figures 7 to 13. In Figure 13, we show

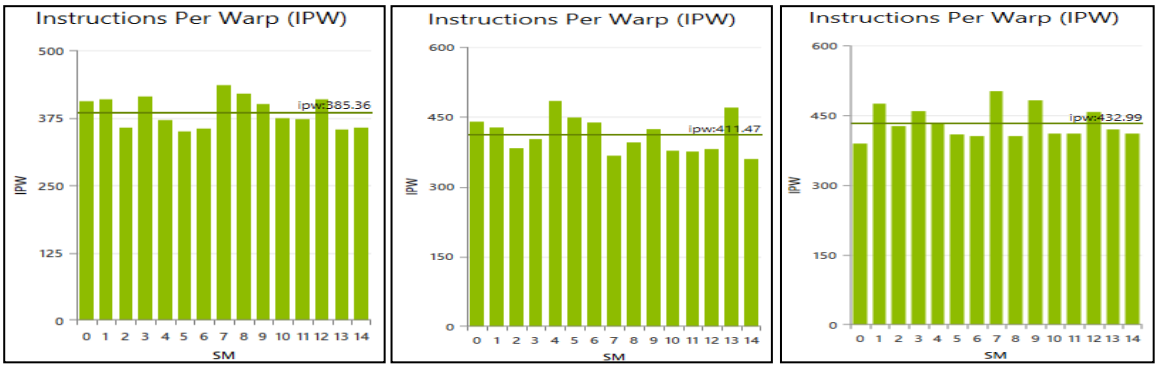


FIGURE 11. Instructions per Warp for AQ: (a) 64M, (b) 128M, and (c) 256M.

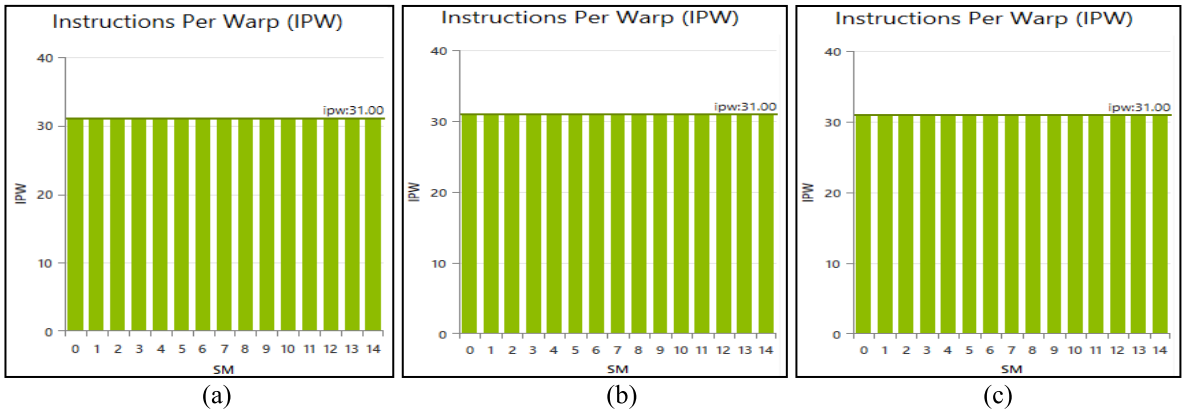


FIGURE 12. Instructions per Warp for BM: (a) 64M, (b) 128M, and (c) 256M.

average results of all the metrics for all the seven datasets. In Figures 7 and 8, results of *SM activity* show that BM has a well-balanced *SM activity* while AQ has some variation in the *SM activity* across each of the 15 SMs throughout the *kernel runtime*. Variation in the *SM activity* across all the SMs leads to *tail effect*, which occurs when some of the SMs finished the workload execution and stay idle while others still busy in executing the workload [18]. The *Tail effect* is one of the reasons of limiting kernel's performance on the device. Though *tail effect* in case of AQ is not very high, in comparison with BM, it is considerably high, which suggests better execution of BM on the GPU.

Figures 9 and 10 show results of *warps launched* metric for three datasets respectively. The results show that BM has launched fewer warps and equal distribution of warps across each SM of the GPU while executing the same workload. The Large variation in the number of *warps launched* across each SM occurs due to lack of enough parallelism in the kernel grid, which is one of the reasons of limiting kernel's performance on the device. As AQ choose *threadsPerBlock* and *blockPerGrid* dynamically at runtime, which may sometimes cause result in the improper kernel configuration, and thus causing variation in *warps launched* metric across all the SMs as shown in Figure 9. On the other hand, in case of BM, optimum values are assigned to variables *threadsPerBlock* and *blockPerGrid* based on the dataset in the source code, which always results in the better kernel configuration, and thus causing well-balanced *warps launched* across all the SMs as shown in Figure 10. Results of *warps launched* also suggest that BM has better execution on the device as variation in *warps launched*.

Figures 11 and 12 provide a comparison of AQ and BM for three datasets based on *instructions per warp* metric. The results show that BM has 31 *instructions per warp* across each SM while AQ has varying and non-uniform *instructions per warp* across each SM in case of all the datasets. As discussed in Section I, BM is an example of data-independent sorting algorithms while AQ is an example of data-dependent sorting algorithms. In data-independent sorting algorithms (such as BM), the algorithms do not change its execution path based on the input but it always follows the same execution path in order to sort the input dataset. On the other hand, in data-dependent sorting algorithms (such as AQ), the algorithm changes its execution path based on the input dataset. This means that BM has a constant set of instructions that it follows in order to sort any dataset regardless of the input dataset size,

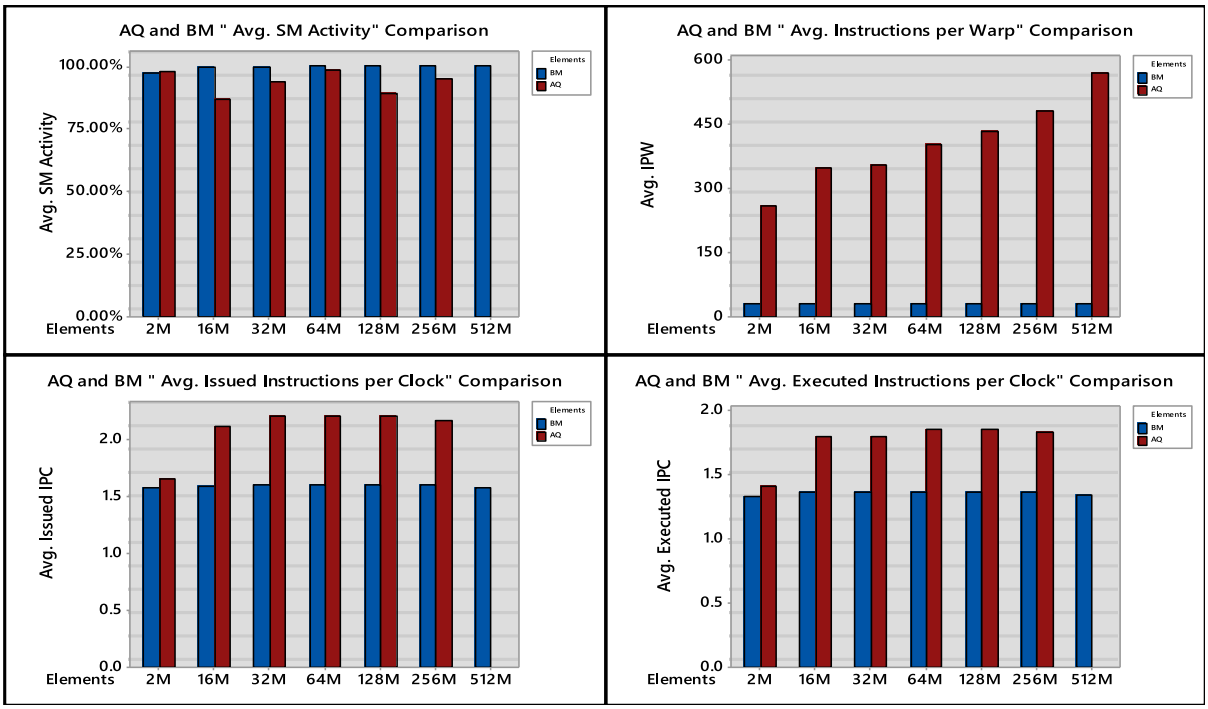


FIGURE 13. AQ and BM instruction statistics: SM activity, IPW, issued IPC, and executed IPC.

while AQ always has a varying set of instructions depending on the size of input dataset, which is depicted in results of *instructions per warp* metric, as shown in Figures 11 and 12. Finally, Figures 13 compares AQ and BM based on average values of all the metrics obtained in *instruction statistics experiment* for all the seven datasets.

It is important to mention that each SM of NVIDIA Tesla K40 GPU contains 4 *warp schedulers* that are able to execute at least one instruction per cycle. At every instruction issue time, each *warp scheduler* selects one warp that is able to make forward progress from its assigned list of warps. For this selected warp, the *warp scheduler* then issues either the next single instruction or the next two instructions. A *warp scheduler* might need to issue an instruction multiple times to actually complete the execution of all 32 threads of a warp. Issuing an instruction multiple times is also referred to as *instruction replay*. In addition, the compute resources required to process the instruction are consumed for every *instruction replay,* which takes away the ability to make forward progress by issuing new instructions on that *warp scheduler* [18]. The results suggest that in case of AQ, there is more *instruction replay* due to which the *warp scheduler* issue instructions multiple times and thus having larger *instructions per warp,* more *warps launched,* and non-uniform *SM activity* across all the SMs as compared to BM. This means that BM has comparatively better execution of the workload across each SM of the GPU as it has well-balanced *SM activity,* fewer *warps launched* and lesser *instructions per warp*. The impact of these metrics on kernel's power and energy efficiency is discussed below.

The *Tail effect* is one of the reasons of limiting kernel's performance on the device that leads to higher latencies that result in higher *kernel runtime*. In case of AQ, *tail effect* results increase in *static power* consumption as the operating temperature rises due to higher *kernel runtime,* and the higher number of *warp launched* across each SM results increase in *dynamic power* consumption of the device due to higher workload execution, thus, affecting the overall *power* and *energy* consumption. The results of *instructions per warp* metric in *instruction statistics experiment* show that BM has 31 *instructions per warp* across each SM while AQ has varying and non-uniform *instructions per warp* across each SM in case of all the datasets. BM is a data-independent sorting algorithm; it does not change its execution path based on the input but it always follows the same execution path in order to sort the input dataset. On the other hand, AQ is data-dependent sorting algorithm; it always changes its execution path based on the input dataset. This means that BM has a constant set of instructions that it follows in order to sort any dataset regardless of the input dataset size, while AQ always has a varying set of instructions depending on the size of input dataset, which is depicted in results of *instructions per warp* metric. Another reason of AQ higher and non-uniform *instructions per warp* is *instruction replay,* due to which an instruction is issued multiple times that results in taking away the ability of a *warp scheduler* to make forward progress by issuing new instructions [18]. The *instruction per warp* metric has a significant effect on the *dynamic power* consumption of the GPU. As modeled by equations 10 and 11, increase in *instructions per warp* results in an increase in *access rate,*

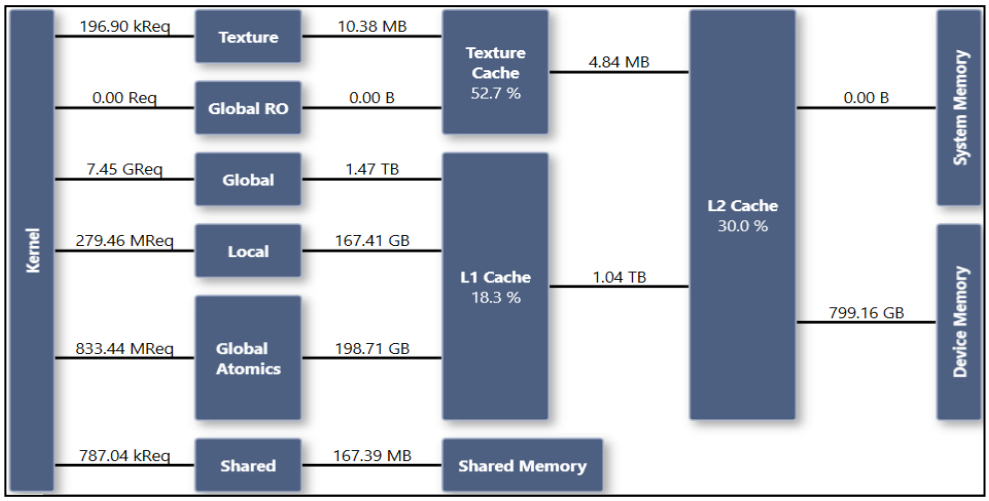


FIGURE 14. Memory statistics experiment overview for AQ: Dataset=512M elements.

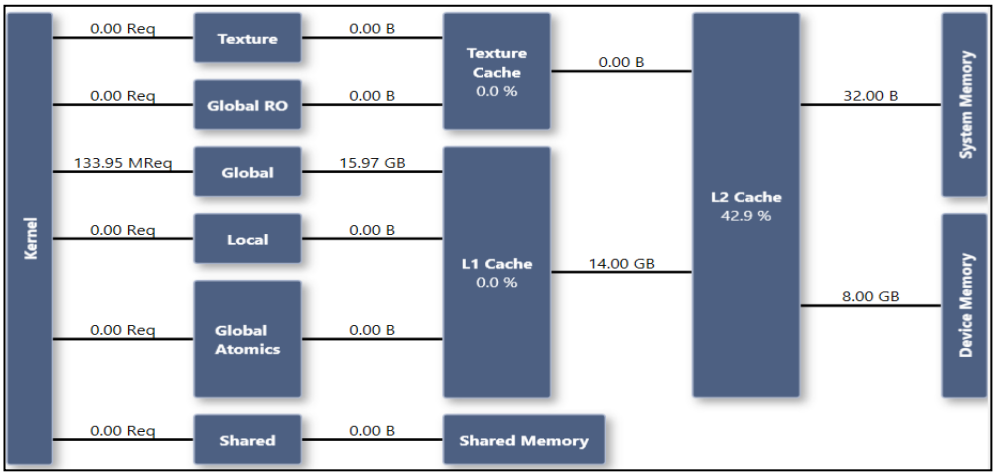


FIGURE 15. Memory statistics experiment overview for BM: Dataset=512M elements.

which shows how often a component is accessed per unit of time. Subsequently, increase in *access rate* has a linear effect on *dynamic power* consumption of the device as shown in [42] and [50]. The results of all the experiments discussed so far are tabulated in Table 4.

Next, we discuss the performance of AQ and BM based on *memory statistics experiment*. From the *memory statistics experiment*, we only show the overview of *memory subsystem* for AQ and BM while executing dataset 512M elements. This highlights the usage of the *memory subsystem* of the K40 GPU for both the algorithms. Figures 14 and 15 show overview of the *memory statistics* experiment for AQ and BM while executing on dataset 512M elements respectively. The nodes in the diagram show various *memory spaces* and the available caches. For the caches, the reported percentage number indicates the cache hit rate; that is the ratio of requests that could be served with data locally available to the cache over all requests made [18]. Links between the nodes in the diagram depict the data paths between the SMs to the *memory spaces*. The results show that AQ uses most of the *memory subsystem* that includes the *global memory, local memory, texture memory, global atomics,* and *shared memory spaces*. On the other hand, BM only uses the *global memory space,* which suggests that BM performs less hard work than AQ because it uses only the *global memory* space, having fewer *data movements* between different *memory spaces* and thus resulting in lower *kernel runtime*. The impact of memory usage on power and energy consumption of kernel is discussed below.

The *memory statistics experiment* provides a very clear explanation of the *power* and *energy* efficiency advantage of BM over AQ. The results of *memory statistics experiments* show that AQ uses most of the *memory subsystem* that includes the *global memory, local memory, texture memory, global atomics, register file,* and *shared memory spaces*. On the other hand, BM only uses the *global memory space.*

Equations 7 and 8 suggest that *dynamic power* consumption of SMs is dependent on the number of components used in the SMs. As AQ uses more components of the SM than BM, thus it has higher *power* and *energy* consumption. On the other hand, equation 9 models the *dynamic power* consumption of the *memory subsystem*, which is dependent on the usage of *global* and *local* memory spaces. Since BM uses only the *global memory space,* thus it consumes less *power* and *energy* than AQ because AQ uses both *global* and *local* memory spaces of the device memory. Using different memory spaces of the device memory in case of AQ means that there will be more data-movements between different components of the device, which is an expensive process in terms of power consumption as has been demonstrated in several studies [52]–[55]. For example in [53], it demonstrated that data-movement of three 64-bit inputs operand between registers and execution unit consumes as much energy as doing a double precision floating point operation. In the same context, it is stated in [46] that ''Dynamic power is reduced by decreasing the access count to each register bank by compressing the operand data into fewer physical register banks. Furthermore, dynamic power is reduced because the compressed register read and write operations activate fewer bitlines in the register file and fewer bits are moved across the wires between register file and execution unit where the data is processed.'' In addition, Connors and Qasem [47] identified that reducing traffic in device memory leads to improvements in both performance and power consumption. Furthermore, the US Department of Energy (DOE) report Architectures and Technology for Extreme Scale Computing [55] states that ''The primary design constraint for future HPC systems will be power consumption.

. . . Data movement will be a bigger factor for system energy consumption and cost than FLOP/s. . . .

Energy and performance costs should be reflected in abstract machine model.''

V. CONCLUSIONS AND FUTURE WORK

Excessive power consumption is one of the major obstacles to achieve exascale performance in a reasonable power budget. Existing techniques are not appropriate for exascale computing. New techniques and solutions are required to reduce the high power requirements of the prospective exascale systems.

In a recent research, we found that evaluating power and energy consumption of fundamental algorithms can offer a reasonable amount of power and energy saving. The hypothesis was that some algorithms can have an inherent algorithmic power and energy consumption advantage depending on the algorithm's fundamental design. We expected BM to be more power and energy efficient because of its fundamental design that is logically suitable for parallel platforms such as GPUs.

In this research, we further investigate BM under various experiments offered by *NVIDIA Nsight Visual Studio* to explore its inherent algorithmic power and energy efficiency. Results of each experiment show how efficiently the algorithm got executed on the GPU while sorting a dataset. It is revealed in the results that a simple in-place BM executes more efficiently than an optimized AQ because BM is logically suitable for the parallel architecture of the GPU. Furthermore, in our results analysis, we discuss the factors that lead to higher *power* and *energy* consumption of GPU that include *data movement*, larger *access rate*, using more components of the SM and the *device memory*, imbalance *SM activity* across all the *streaming multiprocessors (SMs)* that causes *tail effect*, lower and imbalance *achieved occupancy* across all the SMs, lower percentage of *eligible warps*, and larger and imbalance *instructions per warp* across all the SMs that may cause *instruction replay*. It should be noted that some of the factors discussed above depend explicitly on the algorithm's fundamental design, the workload (dataset) executing on the GPU, and the *kernel size*. For instance, *warps launched* metric in *instruction statistics experiment* depends on the workload, i.e. higher workload results in higher *warps launched* and vice versa, but it also depends on the algorithm's design (AQ or BM will have different *warps launched* for a Dataset).

The study open insights for further investigating other fundamental software building blocks such as minimal spanning tree algorithms and binary search algorithms, to identify the algorithmic power and energy consumption advantage in order to provide better recommendations for the upcoming exascale systems.

For future work, the work can be extended to other heterogeneous architectures and some other fundamental algorithms can be investigated for power and energy efficiency. It would also be interesting to develop some analytical methodology for representing power and energy complexities of algorithms.

ACKNOWLEDGMENT

We are also thankful to NVIDIA Corporation for equipment donation for our research.

REFERENCES

[1] (Nov. 2017). *Top500 Supercomputers*. Accessed: Nov. 20, 2017. [Online]. Available: https://www.top500.org/list/2017/11/

[2] M. Wehner, L. Oliker, and J. Shalf, ''A real cloud computer,'' *IEEE Spectr.*, vol. 46, no. 10, pp. 24–29, Oct. 2009.

[3] P. Beckman, ''On the road to exascale,'' *Sci. Comput. World*, vol. 116, pp. 26–28, Feb. 2011.

[4] M. A. Al-Hashimi, O. A. Abulnaja, M. E. Saleh, and M. J. Ikram, ''Evaluating power and energy efficiency of bitonic mergesort on graphics processing unit,'' *IEEE Access*, vol. 5, pp. 16429–16440, 2017.

[5] *Parallel Bitonic Mergesort*. Accessed: Feb. 10, 2016. [Online]. Available: http://www.tools-of-computing.com/tc/CS/Sorts/bitonic_sort.htm

[6] NVIDIA. *CUDA Samples-CUDA Toolkit Documentation*. Accessed: Jan. 3, 2016. [Online]. Available: http://docs.nvidia.com/cuda/cuda-samples/index.html#advanced-quicksort-cuda-dynamic-parallelism

[7] P. Kipfer and R. Westermann, ''Improved GPU sorting,'' in *GPU Gems 2*. Reading, MA, USA: Addison-Wesley, 2005, pp. 733–746.

[8] M. J. Ikram, O. A. Abulnaja, M. E. Saleh, and M. A. Al-Hashimi, ''Measuring power and energy consumption of programs running on kepler GPUs,'' *J. Elect. Syst. Inf. Technol.*, Apr. 2018.

[9] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, ''GPU computing,'' *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[10] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger, "Fast in-place, comparison-based sorting with CUDA: A study with bitonic sort," *Concurrency Comput., Pract. Exper.*, vol. 23, no. 7, pp. 681–693, 2011.

[11] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger, "A novel sorting algorithm for many-core architectures based on adaptive bitonic sort," in *Proc. 26th IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2012, pp. 227–237.

[12] R. Baraglia, G. Capannini, F. M. Nardini, and F. Silvestri, "Sorting using bitonic network with CUDA," in *Proc. 7th Workshop Large-Scale Distrib. Syst. Inf. Retr. (LSDS-IR)*, 2009, pp. 1–8.

[13] NVIDIA. *CUDA Dynamic Parallelism Programming Guide*. Accessed: Nov. 20, 2017. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_Dynamic_Parallelism_Programming_Guide.pdf

[14] C. Luo and R. Suda, "A performance and energy consumption analytical model for GPU," in *Proc. IEEE 9th Int. Conf. Dependable, Autonomic Secure Comput.*, Dec. 2011, pp. 613–620.

[15] J. Coplin and M. Burtscher, "Energy, power, and performance characterization of GPGPU benchmark programs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 1190–1199.

[16] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proc. Int. Green Comput. Conf.*, Aug. 2010, pp. 115–122.

[17] D. Q. Ren and R. Suda, "Investigation on the power efficiency of multi-core and GPU processing element in large scale SIMD computation with CUDA," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 309–316.

[18] NVIDIA. *NVIDIA Nsight Visual Studio Edition*. Accessed: Nov. 20, 2017. [Online]. Available: http://docs.nvidia.com/gameworks/index.html#developertools/desktop/nsight/nsight_visual_studio_edition_user_guide.htm

[19] NVIDIA. Accessed: Nov. 20, 2017. [Online]. Available: http://www.nvidia.com/object/nvidia-kepler.html

[20] (Nov. 2017). *Green500 Supercomputers*. Accessed: Nov. 20, 2017. [Online]. Available: https://www.top500.org/green500/lists/2017/11/

[21] NVIDIA. Accessed: Nov. 20, 2017. [Online]. Available: http://www.nvidia.com/object/nvidia-kepler.html

[22] NVIDIA. Accessed: Nov. 20, 2017. [Online]. Available: http://www.nvidia.com/object/gpu-applications.html

[23] NVIDIA. *CUDA Programming Guide*. Accessed: Nov. 20, 2017. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[24] M. Al-Hashimi, M. Saleh, O. Abulnaja, and N. Aljabri, "Evaluation of control loop statements power efficiency: An experimental study," in *Proc. 9th Int. Conf. Inform. Syst. (INFOS)*, Dec. 2014, pp. 45–48.

[25] M. A. Al-Hashimi, M. E. Saleh, O. A. Abulnaja, and N. Aljabri, "On the power characteristics of mergesort: An empirical study," *J. Elect. Syst. Inf. Technol.*, 2018.

[26] K. O'Brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou, "A survey of power and energy predictive models in HPC systems and applications," *ACM Comput. Surv.*, vol. 50, no. 3, 2017, Art. no. 37.

[27] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU power: A survey of profiling, modeling, and simulation methods," *ACM Comput. Surv.*, vol. 49, no. 3, 2016, Art. no. 41.

[28] E. L. Padoin, L. L. Pilla, M. Castro, P. O. A. Navaux, and J.-F. Méhaut, "Exploration of load balancing thresholds to save energy on iterative applications," in *High Performance Computing*. Cham, Switzerland: Springer, 2017, pp. 76–88.

[29] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of SimplePower: A cycle-accurate energy estimation tool," in *Proc. 37th Design Automat. Conf.*, New York, NY, USA, Jun. 2000, pp. 340–345.

[30] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, "Power aware computing on GPUs," in *Proc. Symp. Appl. Accel. High Perform. Comput. (SAAHPC)*, 2012, pp. 64–73.

[31] I. Zecena, Z. Zong, R. Ge, T. Jin, Z. Chen, and M. Qiu, "Energy consumption analysis of parallel sorting algorithms running on multicore systems," in *Proc. Int. Green Comput. Conf. (IGCC)*, Jun. 2012, pp. 1–6.

[32] Y. Ukidave, A. K. Ziabari, P. Mistry, G. Schirner, and D. Kaeli, "Analyzing power efficiency of optimization techniques and algorithm design methods for applications on heterogeneous platforms," *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 3, pp. 319–334, 2014.

[33] M. Burtscher, I. Zecena, and Z. Zong, "Measuring GPU power with the K20 built-in sensor," in *Proc. Workshop Gen. Purpose Process. Using GPUs*, 2014, pp. 28–36.

[34] J. Coplin and M. Burtscher, "Power characteristics of irregular GPGPU programs," in *Proc. Int. Workshop Green Programm., Comput., Data Process. (GPCDP)*, Dallas, TX, USA, 2014.

[35] E. L. Padoin, L. L. Pilla, F. Z. Boito, R. V. Kassick, P. Velho, and P. O. A. Navaux, "Evaluating application performance and energy consumption on hybrid CPU+GPU architecture," *Cluster Comput.*, vol. 16, no. 3, pp. 511–525, 2013.

[36] S. Roy, A. Rudra, and A. Verma, "Energy aware algorithmic engineering," in *Proc. IEEE 22nd Int. Symp. Modelling, Anal. Simulation Comput. Telecommun. Syst.*, Sep. 2014, pp. 321–330.

[37] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *Proc. 4th Conf. Innov. Theor. Comput. Sci.*, 2013, pp. 283–304.

[38] O. Villa *et al.*, "Scaling the power wall: A path to exascale," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2014, pp. 830–841.

[39] B. Dally, "Power, programmability, and granularity: The challenges of ExaScale computing," in *Proc. IEEE Int. Test Conf.*, Sep. 2011, p. 12.

[40] R. Suda and D. Q. Ren, "Accurate measurements and precise modeling of power dissipation of CUDA kernels toward power optimized high performance CPU-GPU computing," in *Proc. Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Hiroshima, Japan, Dec. 2009, pp. 432–438.

[41] J. Lim, N. B. Lakshminarayana, H. Kim, W. Song, S. Yalamanchili, and W. Sung, "Power modeling for GPU architectures using McPAT," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 19, no. 3, 2014, Art. no. 26.

[42] J. Chen, B. Li, Y. Zhang, L. Peng, and J.-K. Peir, "Tree structured analysis on GPU power study," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, Oct. 2011, pp. 57–64.

[43] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and power analysis of ATI GPU: A statistical approach," in *Proc. 6th IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Jul. 2011, pp. 149–158.

[44] J. Pool, A. Lastra, and M. Singh, "An energy model for graphics processing units," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2010, pp. 409–416.

[45] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, "Warped-compression: Enabling power efficient GPUs through register compression," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 502–514, 2015.

[46] T. Connors and A. Qasem, "Power-performance analysis of metaheuristic search algorithms on the GPU," in *Proc. 6th Int. Green Sustain. Comput. Conf. (IGSC)*, Dec. 2015, pp. 1–6.

[47] J. Coplin and M. Burtscher, "Effects of source-code optimizations on gpu performance and energy consumption," in *Proc. 8th Workshop Gen. Purpose Process. Using GPUs*, 2015, pp. 48–58.

[48] M. Ferro *et al.*, "Analysis of GPU power consumption using internal sensors," in *Proc. 16th WPerformance-Workshop Desempenho Sistemas Comput. Comun.*, 2017, pp. 1698–1711.

[49] NVIDIA. *NVIDIA Kepler Architecture*. Accessed: Mar. 3, 2016. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/Tesla-K40-Active-Board-Spec-BD-06949-001_v03.pdf

[50] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. MICRO*, Dec. 2003, pp. 93–104.

[51] P. Poon and Q. F. Stout, "An optimal time-power tradeoff for sorting on a mesh-connected computer with on-chip optics," *Int. J. Netw. Comput.*, vol. 4, no. 1, pp. 70–87, 2014.

[52] B. Dally, "The future of GPU computing," in *Proc. 22nd Annu. Supercomput. Conf.*, 2009, pp. 1–37.

[53] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep./Oct. 2011.

[54] K. Sankaralingam *et al.*, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *Proc. 30th Annu. Int. Symp. Comput. Archit.*, Jun. 2003, pp. 422–433.

[55] R. Stevens, A. White, and S. Dosanjh, "Scientific grand challenges: Architectures and technology for extreme scale computing report," U.S. Dept. Energy, Washington, DC, USA, Tech. Rep., 2009. [Online]. Available: https://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Arch_tech_grand_challenges_report.pdf

OSAMA AHMED ABULNAJA received the B.S. degree in computer science from King Abdulaziz University (KAU), Jeddah, Saudi Arabia, in 1986, and the M.S. degree in computer science and the Ph.D. degree in engineering (computer science) from the University of Wisconsin-Milwaukee, Milwaukee, WI, USA, in 1990 and 1996, respectively. He is currently a Professor of computer science with KAU, where he is also a member of the HPC Research Group, Department of Computer Science. His research interests are fault tolerance, high-performance computing (HPC), systems performance, and systems programming.

MUHAMMAD JAWAD IKRAM received the B.E. degree in computer systems engineering from the University of Engineering and Technology at Peshawar, Pakistan, in 2010, the M.Sc. degree with Distinction in networks and performance engineering from the University of Bradford, U.K., in 2012, and the Ph.D. degree from the Department of Computer Science, King Abdulaziz University (KAU), Jeddah, Saudi Arabia, in 2018. He is a member of the HPC Research Group, Department of Computer Science, KAU. He is recently appointed as an Assistant Professor of computer science in the Department of Computer Science, International Islamic University, Islamabad, Pakistan. His current research interests include GPU computing, HPC, and performance modeling.

MUHAMMAD ABDULHAMID AL-HASHIMI received the B.S. degree in electrical engineering (electronics and communication) from King Abdulaziz University (KAU), Jeddah, in 1987, and the M.S. and Ph.D. degrees in computer science from Texas A&M University in 1993 and 2000, respectively. He is currently an Assistant Professor with the Department of Computer Science, KAU, where he is a member of the HPC Research Group, Department of Computer Science. He has been the Vice Dean of the development in charge of putting in place quality management systems for bachelor programs. His research interests include processor design and high-performance architectures.

MOSTAFA ELSAYED SALEH received the B.S. degree in computer engineering and the M.S. and Ph.D. degrees in computer engineering from Mansoura University, Egypt, in 1992, 1995, and 2000, respectively. He is currently an Associate Professor with King Abdulaziz University, Jeddah, where he is a member of the HPC Research Group, Department of Computer Science. His research interests are data engineering, semantic Web, and high-performance computing.