IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 14, 2017, accepted July 28, 2017, date of publication August 7, 2017, date of current version September 6, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2736525

Evaluating Power and Energy Efficiency of Bitonic Mergesort on Graphics Processing Unit

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

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

Corresponding author: Muhammad Jawad Ikram (mshahid@stu.kau.edu.sa)

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

ABSTRACT Excessive power consumption is expected to be the major obstacle to achieve exascale performance within a reasonable power budget in the upcoming years. In addition, *graphics processing units (GPUs)* are expected to become a significant ingredient in the pursuit of *exascale computing* due to their fine-grained, highly parallel architecture and advancements in performance and power efficiency. To address the power obstacle of exascale systems, we suggest evaluating power and energy consumption of the fundamental software building blocks. We experimentally investigate *power consumption, energy consumption, and kernel runtime* of Bitonic Mergesort (a promising sort for parallel architectures) under various workloads on NVIDIA K40 GPU. The results show some insights in terms of power and energy consumption advantage of Bitonic Mergesort compared with NVIDIA's Advanced Quicksort (a highly optimized parallel quicksort).

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

I. INTRODUCTION

Exascale computing means to achieve order 10^{18} floating point operations per second (FLOPS) and order 10^{18} bytes of storage, which is 1,000 times the capability of today's petascale platforms [1]. Achieving exascale performance in a reasonable power budget requires innovations and advancements in architecture, software and algorithms. Power consumption, fault tolerance, fault rates, and transforming applications from petascale to exascale systems are some of the major challenges to achieve exascale computing. The likely environment for exascale computing is highly parallel many-cored compute nodes, organized in large systems with higher probability of software and system faults [1]–[4]. Simply scaling current technology raises problems such as communication, power and energy consumption. Observing *high-performance computing* (*HPC*) systems on the Top500 list [5], scaling these machines to exascale would produce machines that consume Gigawatts of power. Providing this amount of power would require a medium size nuclear power plant [6]. A US Defense Advanced Research Projects Agency (DARPA) report estimates a reasonable peak of electrical power; according to which maximum power of future HPC systems must be below 20 Megawatts [4]. To overcome these challenges, huge investment is needed in numerous areas of research and development.

Current HPC techniques are not suitable for the next generation of supercomputing (exascale computing) [4]. We need alternatives that can cope with energy consumption restraints to reach the next scale of HPC systems. We suggest that exploring accurate power and energy consumption of fundamental algorithms can offer new ways to reduce the excessive power requirements of the upcoming exascale systems. Fundamental algorithms such as sorting algorithms are the building blocks of numerous scientific and HPC applications. HPC applications such as N-body simulations [7], high performance sparse matrix-vector multiplication implementations [8], graphics algorithms like Bounding Volume Hierarchy (BVH) construction [9], database operations [10], machine learning algorithms [11] and MapReduce framework implementations [12] are some examples that depend exclusively on sorting algorithms. In addition, visibility sorting is used in computer graphics applications to properly render transparent objects and efficiently exploit acceleration features such as early-z test. Furthermore, sorting is also essential in physics simulations to insert the contributing objects into spatial structures for detecting collision [13].

In general, sorting algorithms can be categorized into two classes: data-driven and data-independent. Practically, data-driven algorithms are faster because they only consider the current input value but this may lead to unexpected performance if the input sequence is already sorted [13]. Quicksort is an example of data-driven algorithms. For an input sequence of n items, quicksort has an average complexity $O(n\log(n))$ that is provably optimal but in worst case, quicksort has $O(n^2)$ complexity that is not acceptable. On the other hand, the data-independent algorithms are free of this discrepancy because these algorithms always take the same path for sorting the input sequence. This results in a completely rigid operation because the points of communication are known in advance. These features make the data-independent algorithms a good choice for GPUs as these algorithms will execute on fragment processors that cannot modify their output location in memory on the basis of input sequence. Bitonic Mergesort (BM) is an example of data-independent sorting algorithms that can also be used as a construction method to build a sorting network. BM is adequate for generic parallel architectures as it can operate in-place, needs lower inter-process communication, and is logically suitable for single-instruction, multiple-data (SIMD) architectures [13]. Such features of BM led us to investigate it for power and energy efficiency. We expect BM to be more power and energy efficient compared to a highly optimized data-driven algorithm.

This paper highlights the need for finding new techniques to address the power obstacle of exascale systems and discusses the critical issues and challenges associated with power and energy consumption of GPU used to execute HPC applications. In this research, we experimentally investigated power and energy consumption of BM [14] and compared it with NVIDIA's Advanced Quicksort (AQ) [15]. NVIDIA Kepler (Tesla K40c) [16] GPU was used as a test platform due to its outstanding high-performance capabilities. Apart from power and energy consumption investigation of BM, we also evaluated its *kernel runtime* and compared it with that of AQ. *Kernel runtime* means the runtime of a kernel that is executing on GPU and as a result the GPU consumes more power than its idle state power. It is different from the *execution time* of a sorting algorithm. *Execution time* of a sorting algorithm means the total time the algorithm takes to sort a dataset in the source code. *Execution time* of a sorting algorithm is based on its *time-complexity* and *space-complexity*, and it can be obtained by calling timestamp functions in the source code. On the other hand, we obtained *kernel runtime* by reading the GPU built-in sensor. It is explained in detail in Section 1V.

Generally, optimized versions of quicksort are good choice for most applications due to their faster *execution times* but our experimental study shows that AQ (a highly optimized quicksort) have longer *kernel runtimes* than BM (a simple in-place bitonic sort) in most cases. The in-place BM has a memory space advantage and is comparable to AQ in this regard. Longer *kernel runtimes* of AQ means that it keeps the GPU busy for longer time than BM while sorting the same dataset even with comparatively smaller *execution times* than BM. Since the next generation of HPC (exascale) is more concerned about power consumption, the power and energy consumption advantage of BM over AQ can open new insights for exascale research. Researchers can start to rethink fundamental algorithms from power and energy perspective to provide better recommendations for the upcoming exascale systems.

The results provide a comprehensive comparison of AQ and BM based on 3 metrics: *peak power*, *energy* and *kernel runtime*, described as follows.

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

In general the contributions of this paper are as follow:

- We present an experimental methodology for measuring power and energy consumption of kernels running on Kepler GPUs.
- We provide a comprehensive comparison of power profiles of AQ and BM.
- We identify power and energy consumption advantage of BM by comparing it with AQ.

The rest of this paper is organized as follows. Section II presents a concise literature review of related research. Section III presents a brief background of algorithms, test platform, NVDIA System Management Interface, and related CUDA basics. In Section IV, experimental setup and methodology is presented. Section V provides a discussion and an evaluation of results. Finally, section VI concludes the work and suggests some future research directions.

II. LITERATURE REVIEW

Vila *et al.* [17] presented research of NVIDIA Corporation towards designing exascale systems by incorporating features addressing the scaling obstacles of performance and energy efficiency. They evaluated several HPC applications on a number of architectural concepts and demonstrated energy improvements obtained from: circuit and packaging innovations such as low-voltage SRAM; low-energy signaling, and on-package memory. They highlighted power and performance predictions for their exascale research architecture and elaborated the scaling of features regarding future process technologies. The researchers concluded that advancements in performance of applications will require more innovations in algorithms and architecture to improve memory locality, better scaling, and integer execution efficiency.

Dally [18] discussed the challenges for exascale computing and recent techniques addressing those challenges. He argued that obtaining exascale performance in a power budget of 20 MW will require a 200-fold improvement in energy per instruction: from 2 nano-joules to 10 pico-joules. He further discussed that programing exascale machine will require more productive parallel programming environments.

Subramaniam *et al.* [19] analyzed current trends in energy efficiency from the Green500 list [20] of supercomputers and estimated expectations for the upcoming years. Specifically, they first provided an explanation of energy efficiency trends in HPC systems from the Green500. They then modeled and forecasted the energy efficiency of next generation HPC systems. Further, they presented exascalar, i.e. a holistic metric to measure distance from the exaflop goal.

Rajovic *et al.* [21] explored an alternative to current supercomputers that builds on low power mobile processors. They presented the world's first ARM based cluster for HPC. They examined energy-efficiency of ARM Cortex-A0 processor, and finally, explored techniques to improve energy efficiency by increasing compute density.

Moullec *et al.* [22] proposed power prediction technique in embedded systems that use multithreaded processors for delivering a feasible solution for implementing wireless embedded applications. Their design objective was to evaluate and provide comparison of power consumption of algorithms that are executed on multithreaded Xinc processor.

Analyzing power consumption of applications on GPUs and other heterogeneous devices has been discussed in [23] and [24]. Nagasaka *et al.* [25] developed statistical models for GPUs based performance counters. Kasichayanula *et al.* [26] carried out a study on power consumption of numerous microbenchmarks executing on GPUs.

Zecena *et al.* [27] conducted performance and energy consumption analysis of three sorting algorithms, which are odd-even sort, shell sort, and quicksort. They implemented iterative methods for odd-even sort and shell sort while for quicksort; they used recursive implementation using OpenMP. They analyzed both serial and parallel versions of these three algorithms using a shared-memory system that contained two quad-core AMD 2380 Opteron processors.

Ukidave *et al.* [28] analyzed power/performance efficiency of various optimization techniques used on heterogeneous platforms. They evaluated the tradeoff between power and performance by evaluating energy consumption of optimization techniques. Their work focused on discrete GPUs, shared memory GPUs, low power system-on-chip (SoC) devices and includes hardware from NVIDIA, Intel and Qualcomm. They identified architectural and algorithmic factors that have effect on power consumption. They demonstrated that algorithms implementing similar fundamental function can perform differently depending on target hardware and application design.

Burtscher *et al.* [29] presented a power and energy measurement methodology of kernels running on K20 GPU with on-board built-in power sensor. They identified a number of anomalies while using on-board power sensor of K20 GPUs. They validated their methodology using multiple systems, GPUs, scenarios and CUDA programs.

Coplin and Burtscher [30] investigated and contrasted power profiles of irregular and regular programs running on K20 GPU. They also studied effects on power profile while changing GPU's core and memory frequencies, using alternate implementations of the same algorithm and changing input to the program. It is depicted in their results that a single average cannot precisely capture the power behavior of irregular programs. They argued that power must be considered as a function of time and it may have to be evaluated again for each input and after each change in the code.

Roy *et al.* [31] argued on energy management, as a building block for design and implementation of algorithms. They identified memory parallelism as a factor, which affects energy consumption for any given algorithm. Their experiments validate the asymptotic energy complexity model [32]. For sorting algorithms, Roy *et al.* [31] identified that the energy consumed by energy optimal layout (8-way parallel) of selection sort performs much better compared to non-optimal (e.g., 1-way parallel). They identified that quicksort and mergesort show reasonable savings over changes in parallelization due to energy awareness.

Al-Hashimi *et al.* [33] investigated power consumption for three control loops, which are for loop, while loop, and the do while loop. They highlighted that fundamental control statements can exhibit different power consumption while performing the same task. They gave predictions for the best control loop statement in terms of power efficiency.

III. BACKGROUND

In this work, NVIDIA Kepler (Tesla K40c) [16] architecture is used as a test platform. In addition, CUDA 7.5 programming standard [34], NVIDIA System Management Interface (NVSMI) [35], a mathematical package and worksheets are used to successfully accomplish experiments on K40c GPU. In this section, we briefly describe the algorithms, NVIDIA Kepler architecture, NVSMI and CUDA basics.

A. THE ALGORITHMS

Both quicksort and mergesort are based on *divide-and-conquer* principles. Quicksort is honored as one of the top 10 algorithms of the 20^{th} century [36]. In [36], the working of quicksort in its simplest form is described as follows. The *pivot* is determined in a *divide* step, which is then used to partition the input sequence A[x ... z] into two subsequences. After portioning, the *pivot* is placed in A[y], where all the elements smaller than or equal to A[y] are placed in subsequence A[x ... y−1] and all the elements greater than A[y] are placed in A[y + 1 ... z]. In the *conquer* step, the same procedure is used for recursively sorting the subsequences until subsequences of length 1 are obtained. On the other hand, mergesort in its simplest form works as follows. Firstly, the list is split into multiple subsequences.

Secondly, each subsequence is sorted and, finally, all subsequences are merged into sorted sequence [36].

In our experiments, we investigated Bitonic Mergesort [14] and Advanced Quicksort [15] for power and energy efficiency. These two algorithms are briefly described next.

1) ADVANCED QUICKSORT (AQ)

Advanced Quicksort (AQ) is the implementation of a highly optimized version of parallel quicksort developed by NVIDIA Corporation using CUDA dynamic parallelism [37]. It is supported on devices with compute capability 3.5 or greater. It works as follows:

- It uses a small-set insertion sort for list size less than or equal to 32 elements.
- It uses a portioning kernel, which separates an input array based on the given *pivot*. The input array is divided into elements less than or equal to pivot and elements greater than *pivot*. Two quicksorts are launched to sort each of these elements.
- A quicksort coordinator is used to launch proper kernels.
- It performs a per-warp quicksort without inter-warp communication. Selection of data is determined by a warp, which then writes data greater than *pivot* to one buffer and data smaler then *pivot* to another buffer. A unique section of the buffer is acquired through atomics.
- A warp finds its section of the data, and then writes all data less than *pivot* to one buffer and all data greater than *pivot* to the other. Atomics are used to get a unique section of the buffer.
- For optimization, multiple chunks are done per warp to increase in-flight load and reduce the instruction overhead.
- Elements that are greater or smaller than *pivot* are counted by each warp. As the count is known to a warp, it updates an atomic counter. If all are less than or equal to *pivot* then the comparison is adjusted, otherwise it will move nothing and iterate forever.

2) BITONIC MERGESORT (BM)

Bitonic Mergesort (BM) is a sorting algorithm that is designed specifically for parallel platforms. This algorithm was proposed by Ken Batcher [38]. BM is also used as a construction technique for developing sorting networks, that consist of $O(n\log^2(n))$ comparators and have a delay of $O(n\log^2(n))$, where n is the number of items to be sorted [39]. BM is one of the most studied algorithms in GPU computing community [13], [40]–[43]. It is suitable for generic parallel architectures as it can work in-place, needs lower inter-process communication, and is logically suitable for SIMD architectures. A bitonic sequence is composed of two subsequences, one monotonically ascending and the other monotonically descending, for example "V" and ‘‘A-frame’’. The CUDA version of in-place BM used in this paper is based on [14], briefly described as:

- It divides the input sequence into two halves.
- It sorts the lower half and upper half into ascending and descending order respectively, thus, resulting in a bitonic sequence.
- It executes a bitonic merge on the sequence that results in a bitonic sequence in each half and all the larger elements in the upper half.
- Each half is recursively bitonically merged until all the elements are sorted.

B. NVIDIA Kepler ARCHITECTURE

NVIDIA Kepler GK110 is a computational workhorse specifically designed to deliver cutting-edge performance with power efficiency and to address the most overwhelming challenges in HPC [44]. GK110 is composed of 7.1 billion transistors and is capable of performing teraflops of integer, single precision, and double precision performance and high memory bandwidth [44]. Table 1 highlights some key features of Tesla K40.

Numerous scientific applications [45] use Kepler GK110 due to its innovative computing technology. SMX, Dynamic Parallelism and Hyper-Q are three distinguished innovations in Kepler GK110 [44]. SMX unit lies at the heart of GK110 GPU, which is the next generation streaming multiprocessor. Its new innovative design allows more space to processing cores than control logic [44]. Through dynamic parallelism [37], GPU threads can create new threads, without CPU involvement, and adapt to its data, which results in eliminating effectively the superfluous back-and-forth communication between the GPU and CPU through nested kernel computations. Hyper-Q allows several CPU cores to accomplish work on a single GPU concurrently, which significantly increases GPU utilization and reduces CPU idle times [44].

Popular Kepler architecture includes K20, K40 and K80. Due to outstanding high-performance capabilities we used the NVIDIA K40c [16] Kepler-based architecture as a platform in our experiments. K40 can be found in a number of top500 [5] and green500 [20] supercomputers.

C. CUDA BASICS

We used CUDA programming standards to leverage the inherent parallel programming capabilities offered by GPUs. Following is a concise description of terminology used in GPU programming based on CUDA to explain key features of Tesla K40 in Table 1.

Host: means the CPU and its memory. In our case, it is the Intel(R) Xeon(R) CPU E5-2640 2.50GHz and its memory.

Device: means the GPU and its memory. In our case, it is the NVIDIA Kepler K40c GPU and its memory.

Kernel: is a function that executes on the GPU. Applications are executed in parallel on GPU as kernels. One kernel is launched at a time by a number of threads. In our case, we have AQ and BM kernels that contain the underlying sorting algorithm source codes and they are launched on K40 GPU.

TABLE 1. Key features of tesla K40 [16].

Specifications	Tesla K40
Chip	GK110B
Compute Capability	3.5
Single Precision FLOP/s	4.29 TeraFLOP/s
Double Precision FLOP/s	1.43 TeraFLOP/s
Shared Memory per Block	48 KiB
Threads per Block	1024
Registers per Block	65536
Grid Dimensions	[2147483647, 65535, 65535]
Block Dimensions	[1024, 1024, 64]
Streaming Multiprocessors	15
Blocks per Multiprocessor	16
Warps per Multiprocessor	64
Threads per Warp	32
Clock Rate	745 MHZ (Base Clock)
Board Power	235 W
Idle Power	20.57 W
Thermal cooling station	Active fan sink
Memory Size	12 GB
Memory Clock	3.0 GHz

Warp: is a group of 32 threads that are consecutively numbered within a thread block.

Barrier: in the source code, 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 [34].

D. NVIDIA SYSTEM MANAGEMENT INTERFACE

NVIDIA System Management Interface (NVDIA-SMI or NVSMI) is a cross-platform utility that is used to monitor and manage activities on NVIDIA GPUs. It is based on NVIDIA Management Library (NVML) [46] C-based library. NVSMI supports devices from NVIDIA Fermi and higher architectures. Users can generate queries to get information about GPU. GPU power, temperature, performance state and information about many other metrics can be obtained through NVSMI. The information can be displayed to stdout or can be written to log files for scripting purposes. More detailed information on NVSMI can be found in [35]. We used NVSMI in our experiments to query the on-board sensor for power measurements of kernels executing on K40c GPU. Our experimental setup and methodology is discussed in detail the next section.

IV. EXPERIMENTAL SETUP AND METHODOLOGY

We used Fujitsu HPC workstation with a dedicated NVIDIA K40c GPU for experiments. The system specifications are given in Table 2. NVIDIA System Management Interface (NVSMI) was used to read GPU built-in sensor and record data to a log file. NVIDIA Kepler GPU come with built-in sensors that can be queried instantaneously using NVIDIA Management Library (NVML) API calls to monitor the current state of the GPU and record values for ECC status information, GPU load, temperature and fan speed,

TABLE 2. System specifications.

System Manufacturer	FUJITSU
System Model	CELSIUS M720 Power
Processor	Intel(R) Xeon(R) CPU E5-2640 2.50GHz, 2494 MHz,
Operating System	Microsoft Windows 7 Professional x64-based
Physical Memory	8 GB

active computational processes, GPU clock rates, and power management [46]. The built-in sensor provides an adequate basis for elaborating on GPU power management as depicted in [27], [29], and [30]. In [29], the power profiles of various kernels, obtained using Kepler K20 built-in sensor, are validated using a power meter. Thus, we did not bother to use a power meter to validate the data obtained through the K40 built-in sensor.

TABLE 3. Datasets of unsigned integer random numbers.

Datasets	Number of Elements
1	$2 = 2^1 = 2$
2	$1M = 2^{20} = 1{,}048{,}576$
3	$2M = 2^{21} = 2{,}097{,}152$
4	$64M = 2^{26} = 67{,}108{,}864$
5	$128M = 2^{27} = 134{,}217{,}728$
6	$256M = 2^{28} = 268{,}435{,}456$
7	$512M = 2^{29} = 536{,}870{,}912$
8	$1G = 2^{30} = 1{,}073{,}741{,}824$

We tested Advanced Quicksort (AQ) and Bitonic Mergesort (BM) on 8 different datasets of unsigned integer random numbers. The 8 datasets include 2 elements, 1 Mega (M) elements, 2M elements, 64M elements, 128M elements, 256M elements, 512M elements and 1 Giga (G) elements, as described in Table 3. It is important to mention that we also tested AQ and BM on some datasets between 2 elements and 1M elements but there was no significant difference in the results. For the dataset of 2M elements, a small difference was observed in the results. In order to observe a clear difference in power and energy consumption of AQ and BM, we selected the next dataset of 64M elements, and after that we doubled the step size until we reached 1G. For generation of unsigned integer random numbers, we used C built-in function rand( ) and constant RAND_MAX, where rand( ) generates pseudorandom numbers and RAND_MAX is the maximum value returned by the rand( ) function.

Figure 1 summarizes the whole experimental procedure. Following is a step-by-step description of the procedure for executing AQ or BM on the above described datasets (2 elements, 1M, 2M, 64M, 128M, 256M, 512M, and 1G elements) one after another.

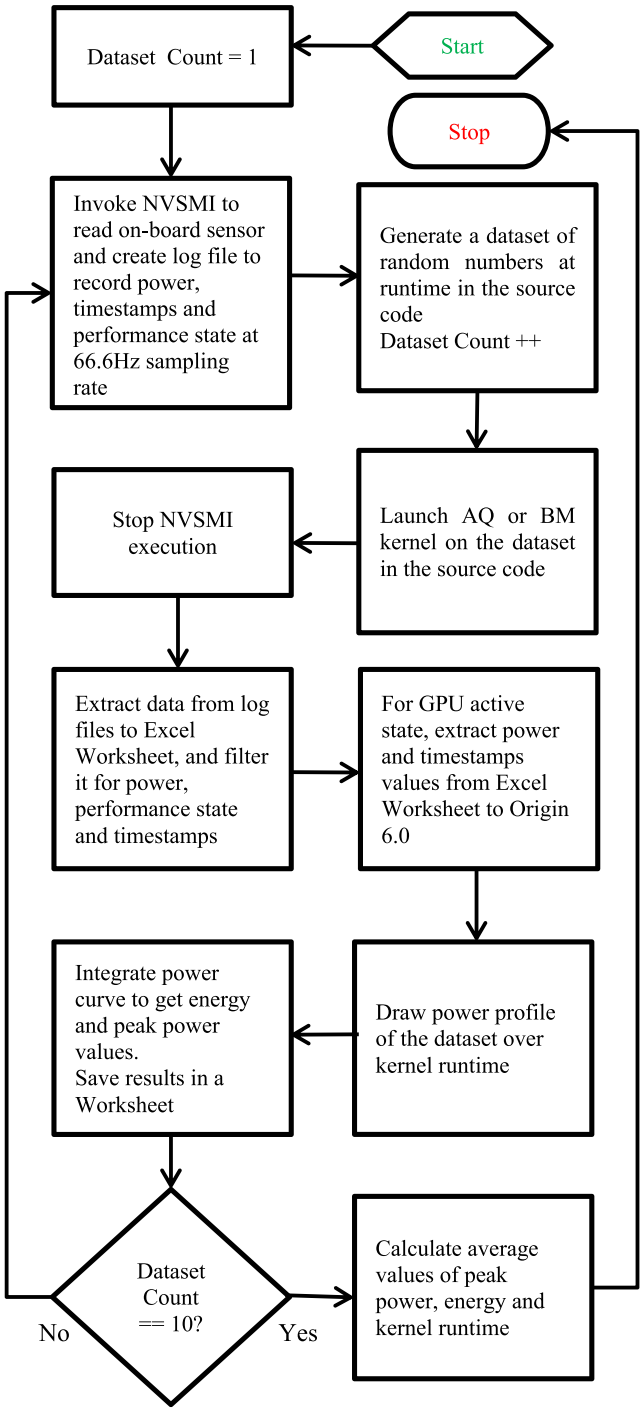

FIGURE 1. Experimental Procedure for Executing AQ and BM on Each Dataset for Measuring Power and Energy Consumption.

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 66.6Hz.
2. A dataset of unsigned integer random numbers to be sorted is generated at runtime in the source code using rand( ) % RAND_MAX.
3. Kernel (AQ or BM) is launched on the dataset in the source code.
4. As the source code gets executed, stop NVSMI execution in the command prompt.
5. Extract data from 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 [47].
7. Obtain power profile of the dataset over *kernel runtime* and record the *peak power*.
8. Using Origin 6.0, integrate power curve to obtain *energy* consumption of the dataset.
9. Repeat step 1 to step 9 for 10 times to compute average values for *peak power, energy* and *kernel runtime*.

We executed AQ and BM separately on each dataset. In order to compute average values (of *peak power, energy* and *kernel runtime*) and increase accuracy in results, AQ and BM were executed on 10 distinct sets of each dataset that were generated randomly at runtime in the source code. This means that the experiment was not limited to only one specific set of each dataset.

Since AQ and BM were executed separately on all 10 sets of the 8 datasets (2 elements, 1M elements, 2M elements, 64M elements, 128M elements, 256M elements, 512M elements and 1G elements), thus, 80 log files were created for AQ and 80 log files for BM using NVSMI. The log files recorded information about power measurements and *performance states* from GPU on-board sensor. Power measurements include the current *power draw* and *power limits* of the GPU board. *Performance states* vary from P0 (maximum performance) to P12 (minimum performance). *Kernel runtime* was calculated based on *timestamps* and *performance states*. We read GPU built-in sensor at a sampling rate of 66.6Hz or in other words, we read the sensor every 15ms and recorded data into a log file. This sampling rate (66.6Hz) results in accurate power measurements as recommended in [29]. Finally, we compared AQ and BM based on *average energy*, *average peak power* and *average kernel runtime.*

Figure 2 to Figure 4 provide further explanation to the experimental procedure. Figure 2 and Figure 3 show full (idle + active) power profiles of GPU while executing BM and AQ on one set of dataset 1G elements, respectively. The power profiles show all power values that are recorded from the on-board GPU sensor over the *kernel runtime* at sampling rate of 66.6Hz. GPU *idle* and *active states* are properly highlighted in the figures. The shaded regions under power curves show GPU state (*active state*) during kernel execution. It should be noted that idle power of the K40 GPU is 20.57W [16], which means the power consumption when there is no kernel running on the GPU. As a kernel starts

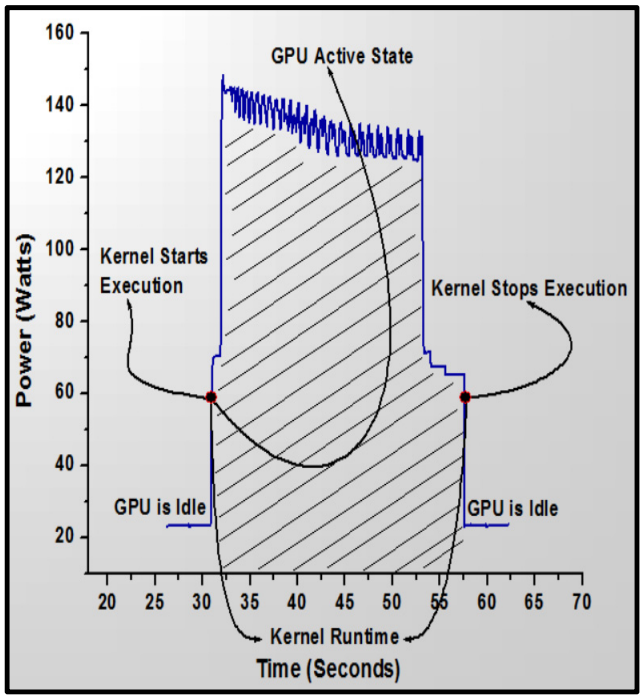

FIGURE 2. GPU Full Power Profile (Idle + Active) for BM: Dataset = 1G Elements.

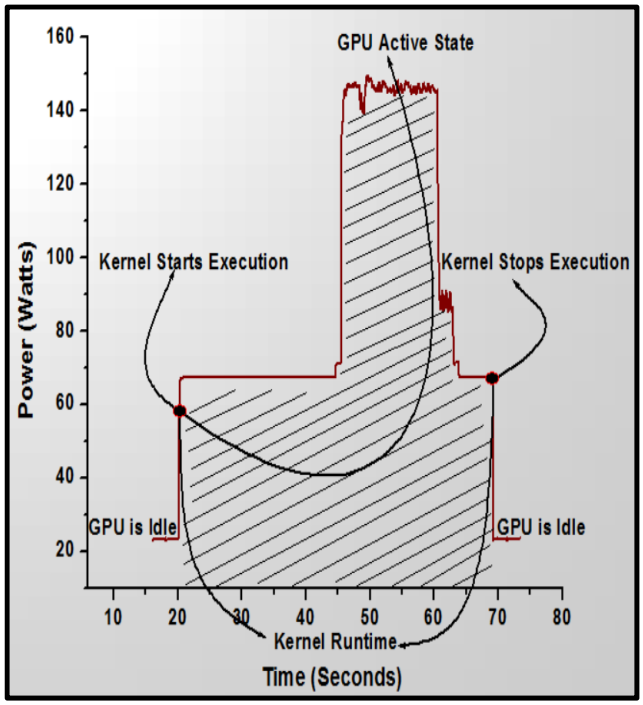

FIGURE 3. GPU Full Power Profile (Idle + Active) for AQ: Dataset = 1G Elements.

execution on the GPU, power consumption of GPU is raised from idle power level. Figure 4 shows a screenshot of a portion of NVSMI log file which shows switching of GPU from *idle* to *active state* (means the kernel starts execution). We can see difference in both the logs, i.e. *performance state*

```
==============NVSMI LOG==============

Timestamp                           : Tue Jun 21 03:08:49 2016
Driver Version                      : 353.90
Attached GPUs                       : 2
GPU 0000:03:00.0
    Performance State               : P8
    Clocks Throttle Reasons
        Idle                        : Active
        Applications Clocks Setting : Not Active
        SW Power Cap                : Not Active
        HW Slowdown                 : Not Active
        Unknown                     : Not Active
    Power Readings
        Power Management            : Supported
        Power Draw                  : 22.88 W
        Power Limit                 : 235.00 W
        Default Power Limit         : 235.00 W
        Enforced Power Limit        : 235.00 W
        Min Power Limit             : 180.00 W
        Max Power Limit             : 235.00 W

==============NVSMI LOG==============
Timestamp                           : Tue Jun 21 03:08:49 2016
Driver Version                      : 353.90
Attached GPUs                       : 2
GPU 0000:03:00.0
    Performance State               : P0
    Clocks Throttle Reasons
        Idle                        : Not Active
        Applications Clocks Setting : Active
        SW Power Cap                : Not Active
        HW Slowdown                 : Not Active
        Unknown                     : Not Active
    Power Readings
        Power Management            : Supported
        Power Draw                  : 49.04 W
        Power Limit                 : 235.00 W
        Default Power Limit         : 235.00 W
        Enforced Power Limit        : 235.00 W
        Min Power Limit             : 180.00 W
        Max Power Limit             : 235.00 W
```

FIGURE 4. Information from NVSMI Log File.

is changed from P8 to P0, *power draw* is changed from 22.88W to 49.04W and *idle flag* is changed to *not active* than *active*. Once the kernel starts execution, GPU power starts increasing based on the dataset and the algorithm running on it. When kernel gets executed, the GPU again comes to *idle state* and *power draw* is changed to *idle state power* again.

Visual Studio 2013 compiler was used to execute the source code, which was configured in *release mode* and *x64* active solution platform. In order to compare AQ and BM under identical configuration, we used same *kernel size* for both AQ and BM kernels. In CUDA, *kernel size* is defined by *block size (threadsPerBlock)* and *grid size (blocksPerGrid)*. In case of AQ, optimum *kernel size* is selected dynamically at runtime that chose 512 *block size* almost in all cases. On the other hand for BM, we used CUDA variables *threadsPerBlock* and *blocksPerGrid* while generating a dataset at runtime. For all datasets greater than 2 elements, we kept 512 *block size* while altered *grid size* based on number of elements in the dataset in case of BM. The number of elements in a dataset were calculated by multiplying *block size* with *grid size* at runtime in the source code. Due to

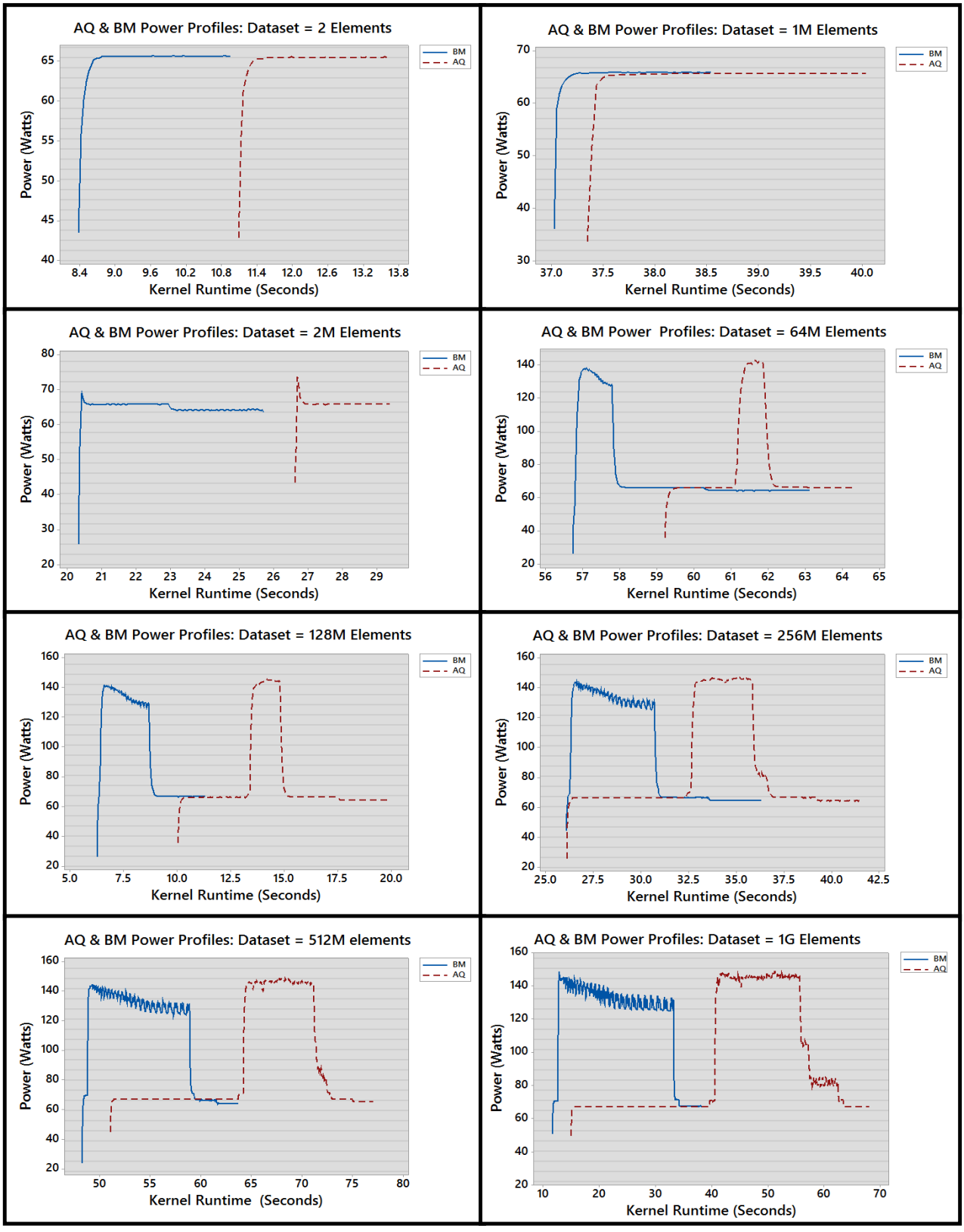

FIGURE 5. AQ and BM Power Profiles while Executing on 1 Set of all 8 Datasets at GPU Active State.

such configuration, we got higher *Achieved Occupancy* and *Streaming Multiprocessor(SM) Activity* of the GPU. *Achieved occupancy* means the ratio of active warps per multiprocessor to the maximum possible active warps [46]. *SM Activity* represents the percentage how long each multiprocessor was active during kernel execution [46]. If a multiprocessor is

executing at least one active warp, it is considered to be active.

Our experimental approach can be used for evaluating the accurate power and energy consumption of any kernels running on Kepler GPUs. It eliminates the use of external power meters and statistical models for power and energy measurements on Kepler GPUs.

A. MEASUREMENT ACCURACY

We took every possible action in order to achieve accuracy in power and energy measurements as given below:

- In order to ensure correct power profiling of AQ and BM kernels while executing on each set of the 8 datasets, we invoked NVSMI to query on-board sensor for the current *power draw* of the GPU and *performance state* prior to execution of the source code in Visual Studio.
- The sampling rate for reading the GPU sensor was kept at 66.6Hz as it is recommended in [29] that this sampling rate results in accurate power profiling.
- *Kernel runtime* was obtained based on the *timestamps* and *performance states* that were recorded to the log files.
- For measuring accurate energy consumption of a GPU kernel (AQ kernel or BM kernel), we did not use the simple approach of averaging power and multiplying it with kernel runtime because this may lead to inaccurate measurements as shown in [29]. Instead, we used integration of power curve over *kernel runtime* to obtain accurate energy consumption of the AQ and BM kernels.

V. RESULTS EVALUATION

Figure 5 shows power profiles of Advanced Quicksort (AQ) and Bitonic Mergesort (BM) during kernel execution on GPU for one set of each of the 8 datasets (2 elements, 1M, 2M, 64M, 128M, 256M, 512M, and 1G elements). Due to space limitations, we only show power profiles for one set of each of the 8 datasets but such power curves (as in Figure 5) were obtained for all 10 sets of the 8 datasets during kernel execution. The figure only shows power profile during GPU *active state* because it represents the actual power consumption of the kernel and it is used for energy calculations. The power profiles show the current *power draw* of the GPU board during execution of AQ and BM kernels. There is no significant difference in power profiles of both AQ and BM while executing on datasets of 2 to 1M elements and thus having the same areas under the power curves. Integration of these curves (2 to 1M elements) over the *kernel runtime* results in almost identical energy consumption. Difference in power profiles is observed very clearly in case of larger datasets, particularly, for dataset of elements greater than 64M, the difference in power profiles is very clear. The power profiles illustrate that in most cases; BM consumes lower power and keeps the GPU less busy than AQ while executing the same workload. Using the power profiles of all 10 sets of the 8 datasets, we obtained *average peak power* and *average energy* consumption of the algorithms. *Average kernel runtime* was calculated based on *timestamps* and *performance states* in the log files.

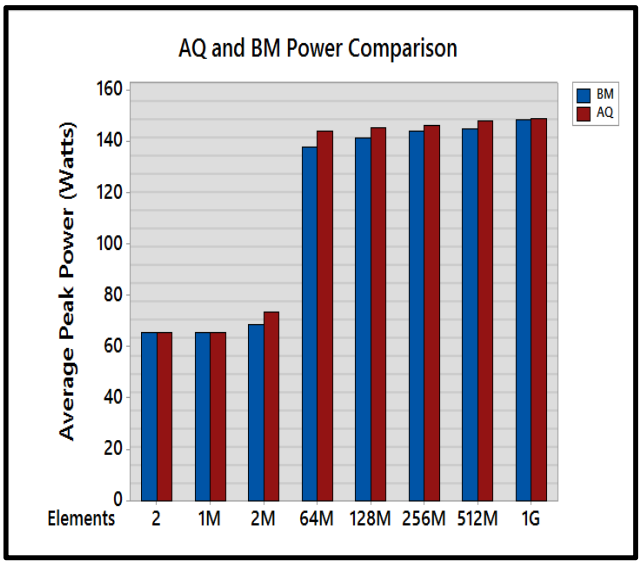

FIGURE 6. AQ and BM Average Peak Power Comparison of all 8 Datasets.

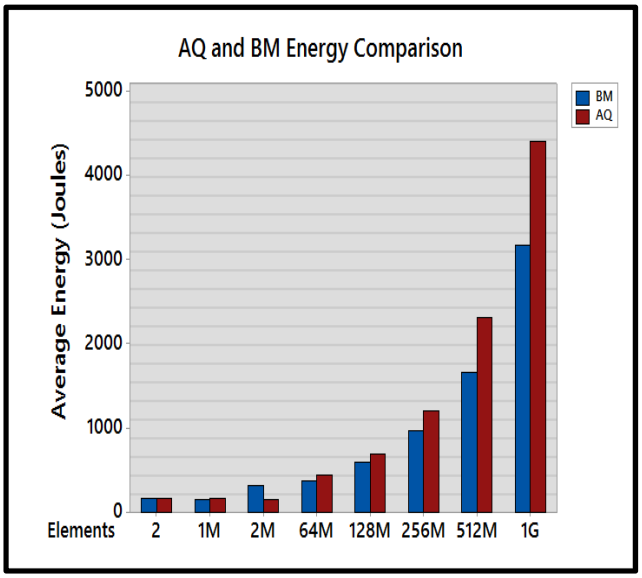

FIGURE 7. AQ and BM Average Energy Comparison of all 8 Datasets.

A comprehensive comparison of AQ and BM *average peak power*, average *energy* and *average kernel runtime* for all 8 datasets (2 elements, 1M, 2M, 64M, 128M, 256M, 512M, and 1G elements) is presented in Figure 6 to Figure 8. It is clear that in most cases, AQ consumes more power and energy than BM because of its higher *kernel runtime*. Higher *kernel runtime* of AQ means that it keeps the GPU busy for longer time than BM while sorting the same dataset. The results reveal that energy consumption of AQ and BM is associated with *kernel runtime* and the area under the power curve of the dataset. It is interesting to notice that in some cases, both AQ and BM have similar *average peak power*

TABLE 4. AQ and BM average peak power, average energy, and average kernel runtime comparison of all 8 datasets.

Datasets	Advanced Quicksort			Bitonic Mergesort			Differences (BM-AQ)		
	Peak P (W)	KRT (S)	E (J)	Peak P (W)	KRT (S)	E (J)	Peak P (W)	KRT (S)	E (J)
2	65.54	2.57	166.31	65.62	2.63	169.90	0.08	0.06	3.59
1M	65.64	2.60	168.86	65.75	2.36	154.17	0.11	-0.24	-14.69
2M	73.61	2.39	156.52	68.64	4.82	312.03	-4.97	2.43	155.51
64M	143.80	5.98	449.50	137.75	4.62	373.82	-6.05	-1.36	-75.68
128M	145.04	8.75	693.19	141.41	6.70	602.68	-3.63	-2.05	-90.51
256M	145.93	14.61	1211.02	143.73	9.88	976.68	-2.20	-4.73	-234.34
512M	148.10	26.11	2311.13	144.71	15.00	1670.28	-3.39	-11.11	-640.85
1G	148.70	48.35	4402.31	148.51	26.62	3172.99	-0.19	-21.73	-1229.32

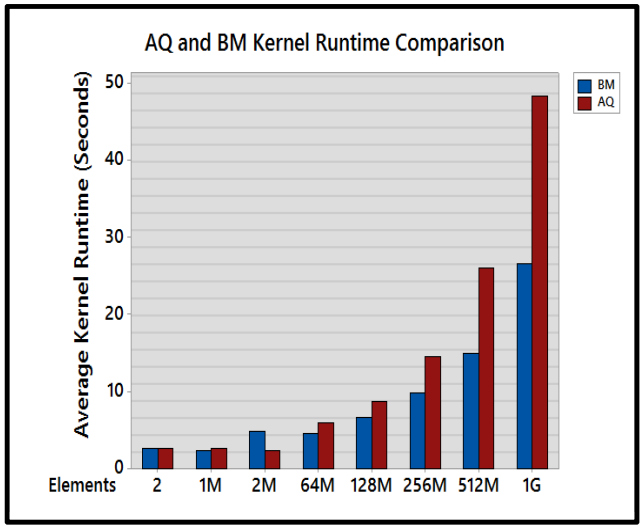

FIGURE 8. AQ and BM Average Kernel Runtime Comparison of all 8 Datasets.

consumption but there is significant difference in *average energy* consumption between both the algorithms. 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. Similarly, for a dataset of 1G elements, *average energy* consumption of AQ and BM is 4402.31J and 3172.99J respectively that means BM has an energy advantage of 1229.32J over AQ on average while sorting a dataset of 1G elements.

Table 4 shows all results that are presented in Figure 6 to Figure 8. The shaded regions in Table 4 show BM advantage over AQ in terms of *average peak power, average energy,* and *average kernel runtime*.

VI. CONCLUSIONS AND FUTURE WORK

The major obstacle to achieve exascale performance under a reasonable power budget is excessive power requirements. There is a need to explore new ways to address that power obstacle. We suggest that exploring power and energy consumption of fundamental algorithms can open new ways to reduce the high power requirements of exascale computing.

In this research, we carried out an experimental study to explore the power and energy efficiency of Bitonic Mergesort (BM) and compared it with NVIDIA's Advanced Quicksort (AQ). We tested both the algorithms on 8 different datasets of unsigned integer random numbers on K40 GPU and obtained average values for *peak power, energy and kernel runtime*. We found that in most cases, BM outperforms AQ in all three metrics. Results suggest that a simple in-place BM has tremendous *energy* and *kernel runtime* advantage, and a reasonable *peak power* advantage over a highly optimized Advanced Quicksort.

The results reveal that there is no significant difference in *peak power, energy* and *kernel runtime* of both AQ and BM while sorting lists of 2 to 1M elements. BM has significantly lower *energy* consumption and *kernel runtime* than AQ in case of larger datasets (greater than 2M elements) while for smaller datasets (less than 2M elements) the difference between both algorithms is not significant. The results also reveal that algorithms having almost similar *peak power* can have significantly different *energy* consumptions due to varying *kernel runtimes*.

Since the major obstacle to achieve exascale performance is excessive power requirements, the power consumption advantage of BM over AQ in particular can open new insights for exascale research. Researchers can start rethinking fundamental algorithms from both power and energy perspective to provide better recommendations for the upcoming exascale systems.

The study also indicates that a huge power and energy advantage can be achieved by properly investigating power and energy consumption of fundamental algorithms.

For future work, we plan to conduct more experiments on BM and compare it with some data-independent sorting algorithms using multiple architectures and programming platforms. This will lead us to identify the underlying algorithmic power and energy advantage of BM. It would also be interesting to investigate other fundamental algorithms such as binary search and minimal spanning tree algorithms for power and energy efficiency. We also plan to develop some analytical methodology for expressing power and energy complexities of fundamental algorithms for generalization.

ACKNOWLEDGMENT

The authors are thankful to NVIDIA Corporation for equipment donation for their research.

REFERENCES

[1] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Commun. ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[2] P. Kogge *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," Univ. Notre Dame, Tech. Rep. TR-2008-13, Sep. 2008.

[3] R. Springmeyer *et al.*, "From petascale to exascale: Eight focus areas of R&D challenges for HPC simulation environments," Lawrence Livermore Nat. Lab., Livermore, CA, USA, Tech. Rep. LLNL-TR-474731, 2011.

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

[5] (Nov. 2016). *Top500 Supercomputers*. [Online]. Available: http://www.top500.org/lists/2016/11/

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

[7] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. Quinn, "Massively parallel cosmological simulations with ChaNGa," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2008, pp. 1–12.

[8] N. Bell and M. Garland, "Implementing sparse matrix -vector multiplication on throughput-oriented processors," in *Proc. Conf. High Perform. Comput. Netw., Storage Anal. (SC)*, 2009, pp. 1–11.

[9] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast BVH construction on GPUs," *Comput. Graph. Forum*, vol. 28, no. 2, pp. 375–384, 2009.

[10] G. Graefe, "Implementing sorting in database systems," *ACM Comput. Surv. (CSUR)*, vol. 38, no. 3, pp. 69–106, 2006.

[11] R. Wu, B. Zhang, and M. Hsu, "Clustering billions of data points using GPUs," in *Proc. Combined Workshops UnConv. High Perform. Comput. Workshop Plus Memory Access Workshop*, 2009, pp. 1–6.

[12] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce framework on graphics processors," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 260–269.

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

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

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

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

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

[18] B. Dally, "Power, programmability, and granularity: The challenges of exascale computing," in *Proc. IEEE Int. Test Conf.*, May 2011, p. 878.

[19] B. Subramaniam, W. Saanders, T. Scogland, and W. Feng, "Trends in energy," in *Proc. Int. Green Comput. Conf. (IGCC)*, 2013, pp. 1–8.

[20] *Green500 Supercomputers*. Accessed on Nov. 1, 2016. [Online]. Available: https://www.green500.org

[21] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez, "The low power architecture approach towards exascale computing," *J. Comput. Sci.*, vol. 4, no. 6, pp. 439–443, 2013.

[22] Y. L. Moullec, C. Leroux, E. Baud, and P. Koch, "Power consumption estimation of the multi-threaded xinc processor," in *Proc. Norchip Conf.*, 2004, pp. 210–213.

[23] 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 Annu. Des. Autom. Conf.*, 2000, pp. 340–345.

[24] R. Suda, "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.*, 2009, pp. 432–438.

[25] 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.*, 2010, pp. 115–122.

[26] 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.

[27] 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)*, 2012, pp. 1–6.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

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

[34] NVIDIA. *CUDA Programming Guide*. Accessed on Oct. 9, 2016. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[35] NVIDIA. *NVIDIA System Management Interface*. Accessed on Jun. 1, 2016. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[36] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Reading, MA, USA: Addison-Wesley, 2011.

[37] NVIDIA. *CUDA Dynamic Parallelism Programming Guide*. Accessed on Jun. 1, 2016. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_Dynamic_Parallelism_Programming_Guide.pdf

[38] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIP Spring Joint Summer Comput. Conf.*, vol. 32. 1968, pp. 307–314.

[39] *Sorting Networks*. Accessed on Feb. 1, 2016. [Online]. Available: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/oddn.htm

[40] 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.

[41] 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.

[42] 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.

[43] 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. 33–40.
[44] NVIDIA. *Kepler Architecture*. Accessed on Jun. 1, 2016. [Online]. Available: http://www.nvidia.com/object/nvidia-kepler.html
[45] NVIDIA. *GPU Applications*. Accessed on Jun. 1, 2016. [Online]. Available: http://www.nvidia.com/object/gpu-applications.html
[46] NVIDIA. *NVIDIA Management Library*. Accessed on Jun. 1, 2016. [Online]. Available: https://developer.nvidia.com/nvidia-management-library-nvml
[47] *OriginLab*. Accessed on May 1, 2016. [Online]. Available: http://www.originlab.com/

MUHAMMAD ABDULHAMID AL-HASHIMI received the B.S. degree in electrical engineering (electronics and communication) from King Abdulaziz University 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, King Abdulaziz University. He has been a Vice Dean of Development in charge of putting in place quality management systems for bachelor programs. His research interests include processor design and high-performance architectures.

OSAMA AHMED ABULNAJA received the B.S. degree in computer science from King Abdulaziz University, Jeddah, Saudi Arabia, in 1986, 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 King Abdulaziz University. His research interests include fault tolerance, high-performance computing, systems performance, and systems programming.

MOSTAFA ELSAYED SALEH received the B.S., 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. His research interests are data engineering, semantic web, and high-performance computing.

MUHAMMAD JAWAD IKRAM received the B.S. degree in computer systems engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2010, and the M.Sc. degree in networks and performance engineering with distinction from the University of Bradford, U.K., in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science, King Abdulaziz University, Jeddah, Saudi Arabia. His current research interests include GPU computing, high-performance computing, and performance modeling.