Performance Evaluation 91 (2015) 255–269

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

A time–energy performance analysis of MapReduce on heterogeneous systems with GPUs

Dumitrel Loghin *, Lavanya Ramapantulu, Oana Barbu, Yong Meng Teo

Department of Computer Science, National University of Singapore, Singapore

ARTICLE INFO

Article history:
Available online 6 July 2015

Keywords:
Heterogeneous systems
GPU
MapReduce
Time–energy performance analysis
Energy efficiency

ABSTRACT

Motivated by the explosion of Big Data analytics, performance improvements in low-power (*wimpy*) systems and the increasing energy efficiency of GPUs, this paper presents a time–energy performance analysis of MapReduce on heterogeneous systems with GPUs. We evaluate the time and energy performance of three MapReduce applications with diverse resource demands on a Hadoop–CUDA framework. As executing these applications on heterogeneous systems with GPUs is challenging, we introduce a novel *lazy processing* technique which requires no modifications to the underlying Hadoop framework. To analyze the impact of heterogeneity, we compare the heterogeneous CPU+GPU with the homogeneous CPU-only execution across three systems with diverse characteristics, (i) a traditional high-performance (*brawny*) Intel i7 system hosting a discrete 640-core Nvidia GPU of the latest Maxwell generation, (ii) a wimpy platform consisting of a quad-core ARM Cortex-A9 hosting the same discrete Maxwell GPU, and (iii) a wimpy platform integrating four ARM Cortex-A15 cores and 192 Nvidia Kepler GPU cores on the same chip. These systems encompass both *intra-node* heterogeneity with discrete GPUs and *intra-chip* heterogeneity with integrated GPUs. Our measurement-based performance analysis highlights the following results. For compute-intensive workloads, the brawny heterogeneous system achieves speedups of up to 2.3 and reduces the energy usage by almost half compared to the brawny homogeneous system. As expected, for applications where data transfers dominate the execution time, heterogeneity exhibits worse time–energy performance compared to homogeneous systems. For such applications, the heterogeneous wimpy A9 system with discrete GPU uses around 14 times the energy of homogeneous A9 system due to both system resource imbalances and high power overhead of the discrete GPU. However, comparing among heterogeneous systems, the wimpy A15 with integrated GPU uses the lowest energy across all workloads. This allows us to establish an execution time equivalence ratio between a single brawny node and multiple wimpy nodes. Based on this equivalence ratio, the wimpy nodes exhibit energy savings of two-thirds while maintaining the same execution time. This result advocates the potential usage of heterogeneous wimpy systems with integrated GPUs for Big Data analytics.

* Corresponding author.
E-mail addresses: dumitrel@comp.nus.edu.sg (D. Loghin), lavanya@comp.nus.edu.sg (L. Ramapantulu), oanabarb@comp.nus.edu.sg (O. Barbu), teoym@comp.nus.edu.sg (Y.M. Teo).

http://dx.doi.org/10.1016/j.peva.2015.06.015
0166-5316/

1. Introduction

The last few years have witnessed Big Data explosion with the increasing volume, velocity and variety of collected data. This explosion is driven by the adoption of data analytics frameworks such as MapReduce [1] and Hadoop [2], its popular open source implementation. MapReduce was originally designed for clusters of homogeneous systems where the scalability is achieved by increasing the number of cluster nodes. But the high power consumption of traditional server nodes and the execution inefficiencies of MapReduce [3] expose the energy issue of Big Data processing.

At the same time with Big Data explosion, systems have become heterogeneous by integrating multiple processing units with different performance-to-power ratios (PPR). This heterogeneity permeates the *intra-chip*, *intra-node* and *inter-node* levels and exposes a large configuration space that could potentially match application's dynamic resource demands. However, efficiently exploiting heterogeneous systems is a daunting task. On the one hand, explicitly programming in C/C++ with OpenMP, OpenCL, CUDA and MPI, and handling fault-tolerance is burdensome. Moreover, adding the energy efficiency as a design goal and choosing the most energy-efficient configuration while meeting an execution time deadline is nontrivial task [4]. On the other hand, frameworks that implicitly handle parallelism, such as MapReduce, are designed for homogeneous systems and may exhibit inefficient execution on heterogeneous systems [3]. To investigate this, we perform a time–energy analysis of MapReduce on *intra-node* and *intra-chip* heterogeneous systems.

Low-power systems, which traditionally target mobile devices market, have made impressive improvements in terms of computational processing and I/O performance. These low-power systems are often called *wimpy* nodes as opposed to the high-performance, *brawny* nodes [5] used in datacenters. With the varying resource demands of mobile apps, these wimpy systems integrate CPU cores with different PPRs, such as ARM big.LITTLE, and accelerators, becoming the main exponents of *intra-chip* heterogeneity. Moreover, the performance improvements of wimpy nodes promote them as an alternative candidate for datacenter computing with the potential to reduce the energy and, thus, datacenter operational costs. Big hardware vendors, such as Dell, AMD, AppliedMicro and Nvidia have already launched server prototypes based on ARM cores [6,7]. However, it remains to be explored if these wimpy systems typically found in smartphones and tablets are suitable for data analytics.

Graphics Processing Units (GPUs) are increasingly being adopted by supercomputers and datacenters due to their immense processing power. For example, the number of Top500 systems with GPUs has increased from less than 10 in 2010 to more than 60 in 2014 [8]. In addition to improvements in performance, GPUs are becoming more energy-efficient. For example, the latest Nvidia Maxwell GPUs are at least two times more energy-efficient compared to the previous Kepler generation [9]. These improvements in the energy efficiency of GPUs and the increasing performance of wimpy systems facilitate their integration. Kayla DevKit is among the first wimpy systems to expose a PCI Express slot to accommodate full-fledged GPUs [6]. Subsequently, Jetson TK1 integrates CPU and GPU cores on the same chip [7], increasing both host-device transfer performance and energy efficiency. With the increasing availability of such intra-node and intra-chip heterogeneous systems with GPUs, it is unclear if they represent a real threat to the traditional homogeneous systems for energy-efficient data analytics.

Motivated by the performance improvements of wimpy systems and GPUs our objective is to investigate the time–energy performance of MapReduce on heterogeneous systems with GPUs. We discuss a measurement-driven comparison across six configurations representing brawny and wimpy systems with both discrete (*intra-node*) and integrated (*intra-chip*) GPUs:

- brawny system with Intel Core i7 CPU [1] (**i7**)
- brawny system with Intel Core i7 CPU and discrete Nvidia Maxwell GPU (**i7**+**GPU**)
- wimpy system with ARM Cortex-A9 CPU (**A9**)
- wimpy system with ARM Cortex-A9 CPU and discrete Nvidia Maxwell GPU (**A9**+**GPU**)
- wimpy system with ARM Cortex-A15 CPU (**A15**)
- wimpy system with ARM Cortex-A15 CPU and integrated Nvidia Kepler GPU (**A15**+**GPU**).

To assess the performance of heterogeneous systems with GPUs, we measure the execution time, power and energy of three MapReduce workloads with diverse resource demands on our Hadoop–CUDA framework. This framework consists of a novel *lazy processing* technique for executing MapReduce applications on heterogeneous systems with GPUs. This technique simplifies application development and requires no modifications to the underlying Hadoop. Additionally, to gain insights into system resource bottlenecks, we profile workload execution and characterize the systems at the CPU, GPU, memory and storage levels. To the best of our knowledge, we are the first to perform such an analysis for heterogeneous wimpy systems with GPUs. Based on this analysis, we make the following key observations:

- For compute-intensive applications, such as BlackScholes, *intra-node* heterogeneity is a boon for brawny systems enabling energy savings of almost 50%, while it is a bane for wimpy systems as they consume more than twice the energy of the homogeneous system.

[1] Although typical brawny server systems employ Intel Xeon processors, we use Intel i7 in our time–energy analysis since it has a better PPR, as shown in the Appendix.

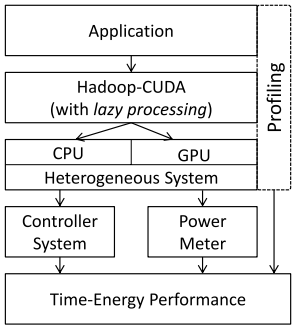


Fig. 1. Methodology overview.

- For applications where data transfers dominate the execution time, such as Grep, heterogeneity exhibits worse time–energy performance compared to homogeneous systems. In the wimpy *intra-node* heterogeneous A9+GPU system, the energy usage of Grep is 14 times higher due to both low host-device transfer bandwidth and high power overhead of the discrete GPU. However, for applications with mixed resource demands, such as Kmeans, *intra-node* heterogeneous systems achieve speedups of up to 1.4 and energy savings of up to 8%.
- As the *intra-chip* heterogeneous A15+GPU system exhibits the lowest energy usage across all applications, we establish an execution time equivalence ratio between a single brawny *intra-node* heterogeneous node and multiple wimpy *intra-chip* heterogeneous nodes. Based on this equivalence ratio, the wimpy systems exhibit energy savings of two-thirds while maintaining the same execution time.

The rest of this paper is organized as follows. We present our methodology in Section 2 and the in-depth time–energy performance analysis of MapReduce workloads in Section 3. We discuss the related work in Section 4 and conclude in Section 5.

2. Methodology

In this section, we describe the methodology of our time–energy performance analysis and present the applications, framework, systems and setup. As shown in Fig. 1, given a MapReduce application that runs on Hadoop, its code is adapted for execution on heterogeneous systems with GPUs. As Hadoop does not support GPU execution, we design a novel Hadoop–CUDA approach to enable executing MapReduce applications on heterogeneous systems with GPUs. To enable a holistic time–energy performance analysis of these applications, we use a power meter to measure power and energy, and we profile their execution at the CPU, GPU, memory and storage levels.

2.1. Applications

We cover financial, machine learning and data mining domains by running the BlackScholes, Kmeans and Grep applications, respectively. BlackScholes (BS) represents a compute-intensive workload that implements a financial model to derive option prices. Given an input file with multiple options and their characteristics, such as stock price, interest rate, expiration time, our program computes the price of each option. Although BlackScholes has been implemented in Hadoop, the code is not available [10]. We adapt the open-source PARSEC 3.0 [11] BlackScholes model implementation which also has an input generator. Map phase applies BlackScholes equations for each option to get the price, and outputs $\langle option\_id,\ price\rangle$ pairs. Reduce phase forwards these pairs to the output. Kmeans (KM) classifies a set of n points in an m-dimensional space into k clusters based on the Euclidean distance between the points and clusters centroids. Kmeans represents a compute-intensive workload with significant memory and I/O demands. Although there is an implementation of Kmeans on Hadoop–CUDA [12], we were unable to execute it on our setup due to the incompatibility between software versions. Consequently, we adapt Mars [13] version which also has an input generator. In this Kmeans implementation, Map phase determines the closest cluster for each point and outputs $\langle cluster\_id,\ point\rangle$ pairs. Reduce phase updates clusters centroids and outputs $\langle cluster\_id,\ centroid\rangle$ pairs. Grep (GR) is a less compute-intensive data mining program which determines input lines that match a regular expression [1]. We translate Hadoop's Java implementation of Grep in C++/CUDA and modify the reduce phase such that it computes the number of lines which match the regular expression. Map phase has the same logic as Hadoop Java version, searching for the regular expression in the input line and outputting $\langle regex,\ 1\rangle$ in case of a match. In our setup, GR searches for string "*the*" in truncated versions of Wikipedia articles dump. To evaluate the adaptability of heterogeneous systems to scale-out workloads, we run the applications with three input sizes as shown in Table 1. These sizes incur execution times ranging from minutes to hours, covering typical data analytics scenarios.

Table 1
MapReduce workloads.

Workload	Input			
	Label	Size [GB]	Description	Source
BlackScholes (BS)	S	0.8	12 million options	PARSEC 3.0 [11]
	M	8.0	120 million options	BlackScholes
	L	24.2	360 million options	Input generator
Kmeans (KM)	S	0.3	$n = 3,474,500$, $m = 34$, $k = 5$	Mars [13]
	M	7.7	$n = 83,388,000$, $m = 34$, $k = 5$	Kmeans
	L	19.3	$n = 208,470,000$, $m = 34$, $k = 5$	Input generator
Grep (GR)	S	0.6	7,800,963 lines	Wikipedia
	M	11.1	166,656,938 lines	Articles dump
	L	22.3	368,789,935 lines	(Truncated)

Algorithm 1 Map phase with lazy processing on GPU

```
static i ← 0

function MAP(key, value)
    buffer[i] ← ⟨key, value⟩
    i ← i + 1
    if i == m then
        process m buffered records on GPU
        emit the results on CPU
        i ← 0
    end if
end function

function CLOSE
    if i > 0 then
        process last i buffered records on GPU
        emit the results on CPU
    end if
end function
```

2.2. Hadoop–CUDA with Lazy Processing

To enable the execution of MapReduce applications on heterogeneous systems with GPUs, we propose a novel *lazy processing* technique and implement it in a Hadoop–CUDA framework. To execute a MapReduce application on the homogeneous systems, we implement it in C/C++ and run it on Hadoop using the Pipes mechanism. In this homogeneous execution, a CPU core sequentially processes each input ⟨*key*, *value*⟩ (record) and outputs zero, one or more ⟨*key*, *value*⟩ pairs. In contrast, our *lazy processing* technique buffers m⟨*key*, *value*⟩ pairs and sends them for processing on m CUDA threads, as shown in Algorithm 1. At the end of each Map task, Hadoop calls *close()* function and the remaining records are sent to the GPU. Using this approach, each of the m CUDA threads processes one record, in contrast to a naive approach of dividing the processing of each record among m CUDA threads. Thus, *lazy processing* ensures that each thread has sufficient work to perform. Moreover, this technique simplifies application development by providing a well-defined implementation pattern and requires no modification of Hadoop framework since it targets only the application code. However, the developer has to select the number of CUDA threads that achieve good performance.

2.3. Heterogeneous systems

To analyze the time–energy performance of MapReduce applications, we use both brawny and wimpy systems representing *intra-node* and *intra-chip* heterogeneity. While the scalability of MapReduce is well studied [1,14,15], our work focuses on the time–energy performance of single-node heterogeneous systems with GPUs. To the best of our knowledge, we are the first to provide an energy usage analysis of MapReduce on heterogeneous wimpy systems with GPUs.

We investigate the time–energy performance of six platform configurations exposed by three heterogeneous systems with diverse characteristics, as shown in Table 2. First, we use a traditional brawny *intra-node* heterogeneous system for comparison with the wimpy systems. This brawny system is based on a 4-core Intel Core i7 processor and 16 GB of RAM. A 512 GB solid-state drive (SSD) is used to store all datasets and workloads, while Ubuntu 13.04 with Linux kernel 3.11.0 is installed on another SSD.

Table 2
Heterogeneous systems with GPUs.

Specifications		i7 (brawny)	A9 (wimpy)	A15 (wimpy)
CPU	Type	Intel Core i7	Nvidia Tegra 3 (ARM Cortex-A9)	Nvidia Tegra K1 (ARM Cortex-A15)
	ISA	x86-64	ARMv7	ARMv7
	Cores	4 (8 threads)	4	4
	Frequency [GHz]	1.60–3.40	0.05–1.40	0.05–2.32
	L1 cache (per core)	32 kB	32 kB	32 kB
	L2 cache	1 MB	1 MB	2 MB
	L3 cache	8 MB	N/A	N/A
GPU	Type	Nvidia GTX 750 Ti		Nvidia GK20A
	Architecture	Maxwell		Kepler
	Cores	640		192
	Memory	2 GB GDDR5		2 GB (shared)
Memory		16 GB DDR3	2 GB LPDDR2	2 GB LPDDR3
Storage	Device	512 GB SSD		
	Port	SATA 3	SATA 2	SATA 3
Networking		Gigabit Ethernet		
Software	OS	Linux 3.11.0	Linux 3.1.10-carma	Linux3.10.40
	Compiler	gcc 4.8.1	gcc 4.6.3	gcc 4.8.2
	Java	jdk1.8.0		
	CUDA	toolkit 6.5		

Second, an *intra-node* heterogeneous wimpy system is represented by a Kayla DevKit equipped with Nvidia Tegra 3 System-on-a-Chip (SoC) having four ARM Cortex-A9 cores and 2 GB of low-power DDR2. This system has a PCI Express x16 port that can accommodate a full-fledged *discrete* GPU. Moreover, it has a SATA interface which enables the connection of a high-capacity disk. We use the same 512 GB SSD to store the datasets and workloads. By default, Ubuntu 12.04 with Linux kernel 3.1.10 is installed on system's flash storage. On top of this OS, we install CUDA toolkit 6.5 and necessary Nvidia drivers.

We use the latest Nvidia Maxwell architecture by hosting on both the wimpy A9 and brawny i7 systems a GTX 750 Ti video card consisting of 640 cores with CUDA compute capability 5.0, and 2 GB of GDDR5 memory. Compared to previous Kepler architecture, Maxwell is two times more energy-efficient [9], being the suitable choice for connecting to a low-power system.

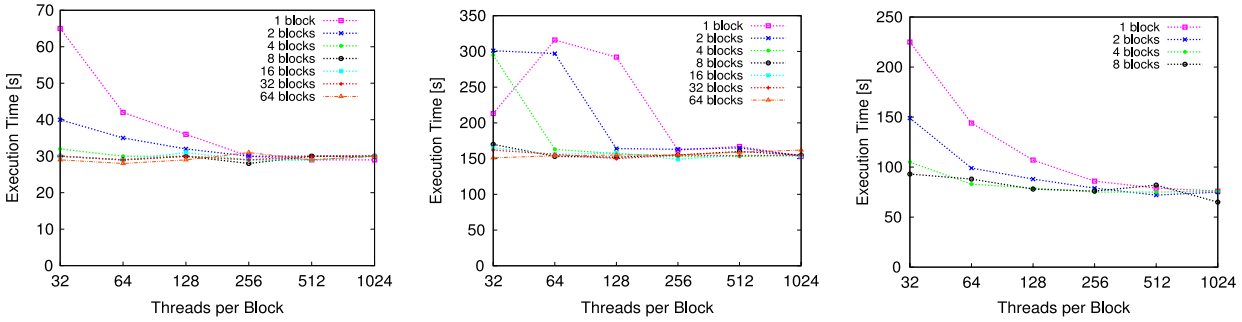
Third, with the increasing adoption of *integrated* CPU–GPU systems [7], we use an *intra-chip* heterogeneous wimpy system represented by Nvidia Jetson TK1 based on Tegra K1 SoC which integrates four ARM Cortex-A15 CPU cores, 192 Nvidia Kepler GPU cores and a shared 2 GB low-power memory. Beside the four fully-fledged Cortex-A15 cores, Tegra K1 incorporates a transparent low-power *companion* core which runs the OS at low system utilization. We connect the same 512 GB SSD to this platform, while the Ubuntu 14.04 OS with Linux kernel 3.10.40 is installed on a 16 GB eMMC. By default, Jetson TK1 comes with CUDA toolkit 6.0 and associated Nvidia drivers. However, we encountered kernel panics and errors when running Hadoop–CUDA on this default setup. Upgrading the drivers and CUDA toolkit to version 6.5 solved these issues.

2.4. Setup

For our performance analysis, we execute the workloads on Hadoop 1.2.1 running on top of Java *jdk1.8.0*. Hadoop is configured to use four map slots and one reduce slot. However, on the wimpy A9 system, workloads with large inputs encounter failures due to insufficient memory. To avoid such failures and to reduce Hadoop's memory usage, we set `io.sort.mb` to 40 and `io.sort.spill.percent` to 0.50, compared to default values of 100 and 0.80, respectively. The native binaries for CPU-only and CPU+GPU are compiled using the *gcc* available on Ubuntu OS and the *nvcc* from CUDA toolkit, and executed through Hadoop Pipes mechanism. The C/C++ code is compiled with *gcc* using maximum level of optimizations (`-O3`). On the wimpy systems, we optimize the code for Cortex-A9 using `-mcpu=cortex-a9 -mtune=cortex-a9`, and Cortex-A15 using `-mcpu=cortex-a15 -mtune=cortex-a15`. Moreover, we enable fast NEON floating-point instructions (`-mfpu=neon`) available on ARM processors.

Performance metrics, such as instructions per cycle (IPC), utilization and memory operations, are collected using *perf* from Linux tools for CPU and *nvprof* from CUDA toolkit for GPU. For each execution, *perf stat* is attached to Hadoop TaskTracker daemon to collect performance counters for spawned Map and Reduce tasks. In contrast, *nvprof* is executed through Hadoop Pipes as a wrapper over CPU+GPU binaries. Energy and power are measured using a Yokogawa WT210 power monitor connected to the AC line of each system. Hence, power and energy values include the inefficiencies of the power adaptor, but since these values are reflected in the energy bill at the end of the month, we believe they are more meaningful than DC values. The logs are collected by a controller system sharing the same LAN as the systems under test.

Each workload is executed three times and average time and energy values are reported, while error bars on the plots indicate the standard deviation. We check workloads outputs for correctness across all input sizes and platform



(a) i7+GPU. (b) A9+GPU. (c) A15+GPU.

Fig. 2. Determining the number of CUDA threads for KM.S.

configurations. Interestingly, we observed that BS outputs are slightly different on the wimpy-only systems. This happens because ARM NEON floating point unit does not fully adhere to the IEEE 754 standard [16]. Thus, when the code is compiled with `-mfpu=vfpv3`, the results are identical with the references, but the execution time increases. Since we want the best performance and the precision loss is acceptable, we report the results obtained by faster NEON instructions.

2.4.1. Determining the number of CUDA threads

Next, we determine the number of CUDA threads, m, that achieves good performance for workloads running on Hadoop–CUDA using the *lazy processing* technique. We empirically determine this number by varying the number of CUDA blocks and threads per block for all workloads using the small datasets. Firstly, we vary the number of threads per block from 32 to 1024 because (i) 32 is the minimum number of threads that are scheduled together, also known as a *warp* in CUDA terminology, and (ii) 1024 is the maximum number of threads per block supported by our GPUs. Secondly, the maximum number of blocks is based on the capabilities of the GPU. For example, we use a maximum of 8 blocks on the less-powerful GPU of the A15 system.

The smallest number of CUDA threads should be selected because it directly affects the size of buffered data. A large amount of buffered data may trigger failures, especially on the wimpy systems with small memories. For all systems, a small number of CUDA threads under-utilizes the GPU and leads to high execution times, as shown in Fig. 2 for KM. When increasing the number of blocks and threads per block, the execution times converge. To select a good configuration from the convergence zone, we execute the workloads with medium datasets. For the systems with discrete Maxwell GPU, we select 16384 CUDA threads representing 16 blocks with 1024 threads per block as a good configuration. For the less-powerful, integrated GK20A GPU we select 2048 CUDA threads split into 8 blocks with 256 threads per block.

3. Performance analysis

In this section, we discuss the results of MapReduce time–energy performance and present an in-depth analysis based on workload execution profiling and system characterization at the CPU, GPU, memory and storage levels. Using these results, we analyze system bottlenecks and discuss the energy efficiency of MapReduce on *intra-chip* heterogeneous wimpy systems.

3.1. MapReduce time–energy performance

We present the time–energy performance of MapReduce workloads on the six platform configurations in Fig. 3 using log scale. The standard deviation among multiple runs is very small, as shown by the error bars. For compute-intensive BS, the GPU significantly improves the execution time only on the brawny system. The speedup of 2.3 leads to 45% energy savings, although the average power of the i7+GPU is slightly higher compared to the i7 CPU-only. On the wimpy A9 system, the speedup is less than 1.1, while on A15 the GPU degrades the execution time by almost 10%. This is a surprising result since BS is a compute-intensive workload, suitable for GPU processing. By further analyzing the behavior of BS on A15, we discover that the compiler optimizations presented in Section 2.4 lead to a speedup of 3.3 compared to the non-optimized binary. As BS consists of a loop in which the option price is computed, it is suitable for loop optimizations, such as loop unrolling, that can significantly improve the execution time. However, for the other two workloads, these compiler optimizations lead to no execution time improvements. For KM, both wimpy systems with GPU exhibit speedups close to 1.2. While the speedup on A15+GPU leads to energy savings of around 20%, on A9+GPU the energy is 80% higher due to the much higher power consumption of the discrete GPU. On the brawny system with GPU, the time improvements are canceled by the higher power consumption of the system with GPU. Thus, the i7+GPU exhibits energy savings of only 8% for KM. For GR, the usage of GPU always results in worse execution time compared to the CPU-only execution. This, corroborated with the higher power consumption of GPU, leads to energy usages that can be even 14 times higher in case of the A9+GPU system, compared to the A9-only. This is because GR is less compute-intensive and host-device transfers cannot be overlapped by

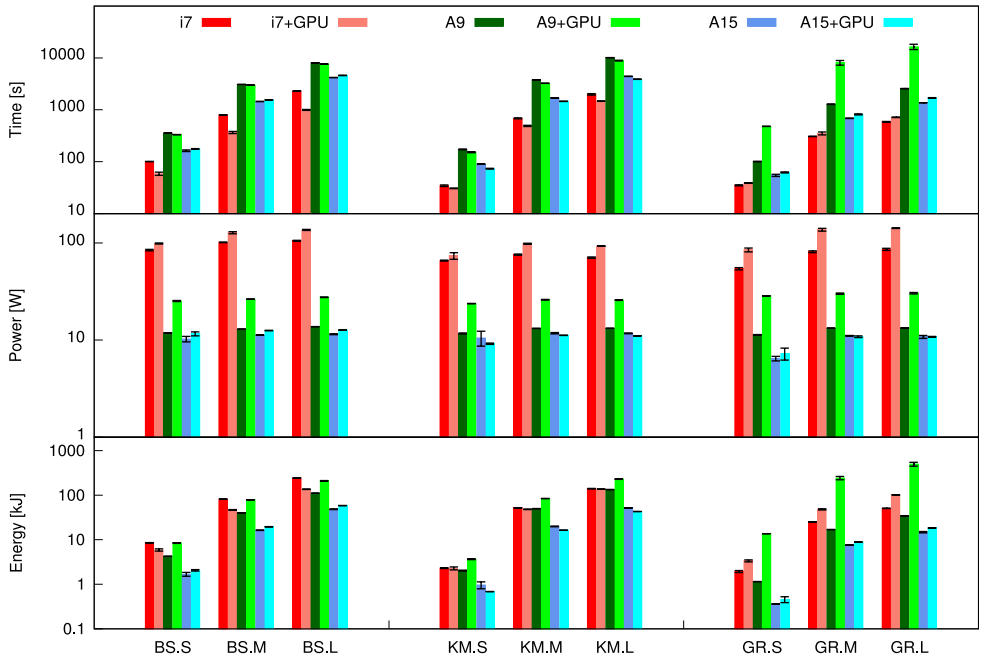
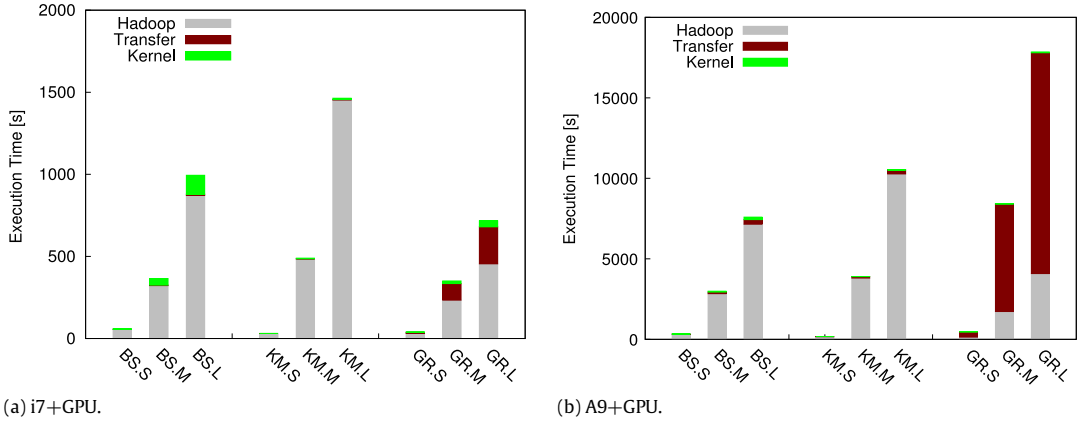


Fig. 3. MapReduce time–energy performance.



(a) i7+GPU. (b) A9+GPU.

Fig. 4. Execution time breakdown for MapReduce on systems with discrete GPUs.

the fast ⟨*key*, *value*⟩ processing. Conversely, for BS and KM there is more processing to be done for each ⟨*key*, *value*⟩ pair and the transfer time is amortized. Based on our analysis of Hadoop–CUDA logs, we compute the proportion of time spent in Hadoop, data transfers to and from GPU and CUDA kernel, as shown in Fig. 4. For GR, the total time spent in host-device transfers is up to 30% and 85% of total execution time on i7 +GPU and A9+GPU, respectively. In contrast, for BS and KM, this time is around 1% and 4%, respectively. The prohibitively large host-device transfer times make workloads such as GR unsuitable for execution on heterogeneous systems with GPUs.

3.2. MapReduce execution profiling

For an in-depth analysis, we profile the entire execution of MapReduce workloads at the CPU, GPU, memory and storage levels. For CPU, we collect hardware counter values such as instructions, cycles, stall cycles, page faults using *perf* Linux tool attached to the TaskTracker daemon of Hadoop. We use *nvprof* to collect GPU execution metrics such as warp execution efficiency, and *dstat* tool available in Linux to get memory and storage usage.

Fig. 5 shows that the memory utilization of MapReduce is not only proportional to the input size, but is also affected by the amount of intermediate data generated by Map phase. For example, KM generates a large amount of intermediate ⟨*key*, *value*⟩ pairs and, hence, it uses around 90% of the memory on all systems when run with M and L datasets. Moreover, this is reflected in the amount of data transferred to and from the storage since Hadoop spills some intermediate data on the

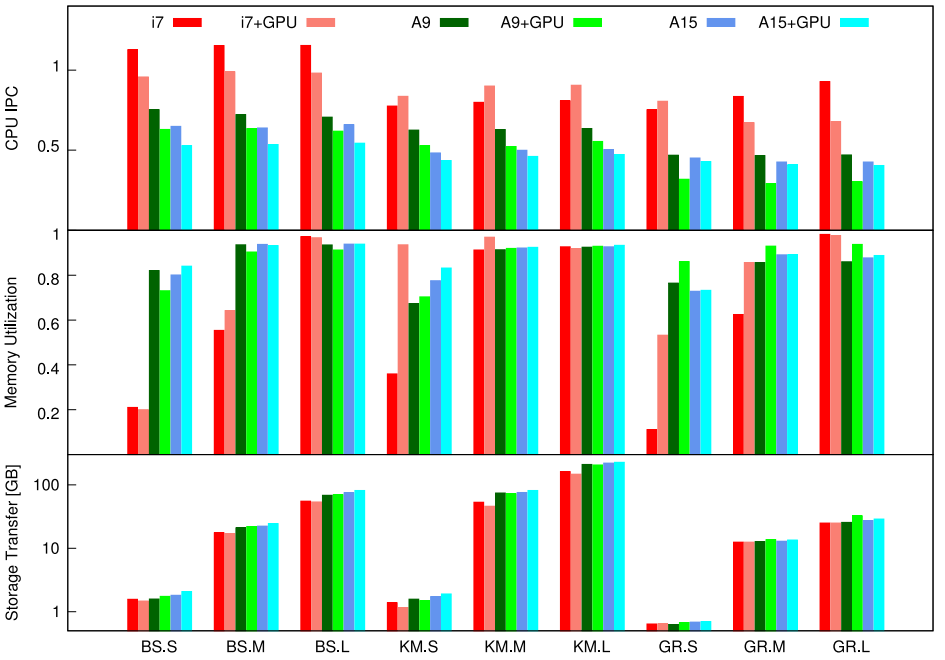


Fig. 5. MapReduce performance at CPU, memory and storage levels.

disk. In case of KM.L, the input size is around 19 GB but the total amount of data moved to and from the storage is more than 100 GB. In contrast, the Map phase of GR generates small ⟨*key*, *value*⟩ pairs only for matching input records, hence, GR has lower memory and storage utilizations. This high memory utilization exposes the limitations of the wimpy systems where Hadoop is constrained by the small memory size. For the same workload, Hadoop uses more than 90% of the 16 GB available on our brawny system, while on the wimpy systems it can use a maximum of 2 GB. Thus, during MapReduce execution on the wimpy systems, the storage is used by the OS virtual memory and by Hadoop spill mechanism. Consequently, there are more page faults and storage transferred bytes for these systems compared to the brawny one, as depicted in Fig. 5. This fact is more visible on the A15+GPU system where the same small memory is shared between CPU and GPU. For example, KM.L on A15 +GPU exhibits 1.75 times more page faults and transfers 33 GB more data to the storage compared to i7+GPU.

Next, we analyze the IPC as a metric for CPU efficiency. Consistent with our workload description, BS and GR have the highest and the lowest IPC, respectively. Moreover, CPU+GPU has lower IPC compared to CPU-only execution because the most compute-intensive part of the workload is offloaded to GPU while the CPU executes Hadoop housekeeping. But the brawny system is the exception since (i) GR has higher IPC compared to KM and (ii) KM has higher IPC on CPU+GPU than on CPU-only. Firstly, both KM and GR have high memory utilization on a system with high clock frequency, such as i7. At higher clock frequencies, the speed gap between CPU and memory is larger and more CPU cycles are wasted waiting for memory requests to be serviced [4]. These wasted cycles are reported by *perf* as *stall cycles*. For example, BS.L, KM.L and GR.L exhibit 27%, 43% and 39% stall cycles, respectively, as reported to total cycles. These values show a strong correlation with the IPC. Secondly, KM has higher IPC on CPU+GPU than on CPU-only because the workload exhibiting stall cycles is offloaded to the GPU. Among systems, i7 exhibits the highest IPC since it has a deeper pipeline and bigger issue width. Surprisingly, A9 system has higher IPC than A15 while performing poorer in terms of execution time. We analyze this result based on the fact that the execution time is determined by the total number of executed instructions divided by IPC, clock frequency and utilization,

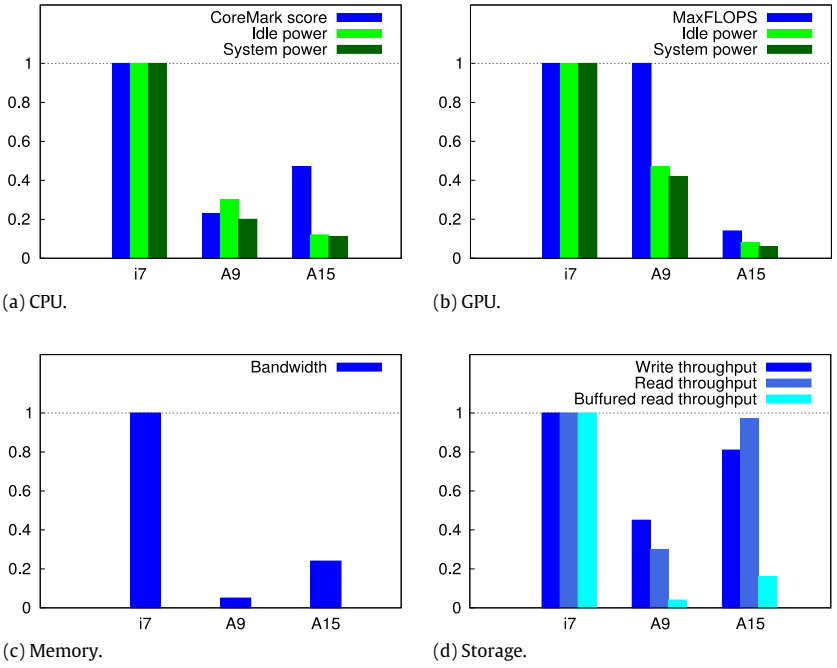
$$T = \frac{Instructions}{IPC \cdot f \cdot U}. \tag{1}$$

A9 and A15 *perf* logs show that the number of instructions and the utilization are similar. But the clock frequency of A15 is almost two times higher, as indicated in Table 2, hence, there are more stall cycles due to memory requests. Our measurements indeed show that A15 executes more cycles than A9 and, thus, has a lower IPC. However, this lower IPC is overcome by the increased frequency, thus, leading to a better execution time.

Lastly, we analyze the execution of the three workloads on GPU using the *warp execution efficiency* metric exposed by *nvprof*. This metric represents the average active threads per warp divided by the maximum number of threads per warp, which is 32 for both GPUs used in our evaluation. CUDA threads in a warp are inactive if they wait for other threads to execute a divergent path or they finished the execution. As expected, GR has the highest proportion of such inactive threads which decrease the efficiency to 17%–26%, as shown in Table 3. In contrast, BS and KM achieve 61%–99% warp execution efficiency.

Table 3
GPU profiling.

Workload	Warp execution efficiency [%]		
	i7+GPU	A9+GPU	A15+GPU
BS	97	97	77
KM	99	98	61
GR	26	26	17



(a) CPU. (b) GPU.

(c) Memory. (d) Storage.

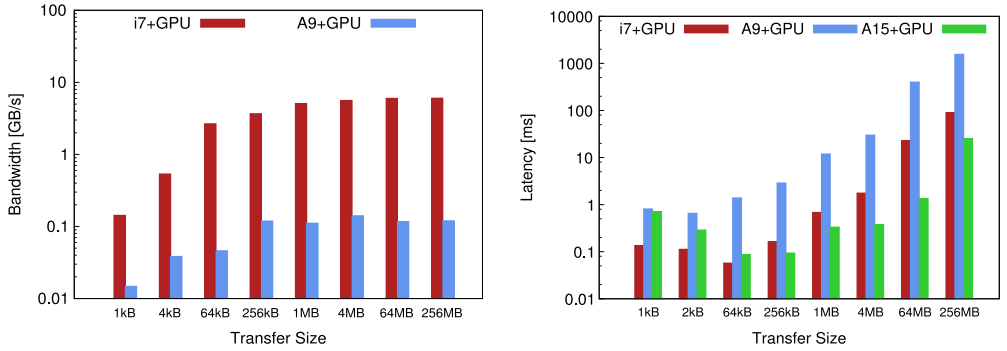
Fig. 6. Systems characterization.

In summary, we draw the following observations based on our profiling of MapReduce execution. First, our workload description is validated since BS is the most compute-intensive workload and GR is the least compute-intensive workload. Second, the small memory of wimpy systems hinders MapReduce since Hadoop has to use the storage to spill intermediate data.

3.3. Heterogeneous systems characterization

To augment the explanation of MapReduce performance results, we characterize the systems[2] at the CPU, GPU, memory and storage levels, as shown in Fig. 6. To assess CPU performance, we run CoreMark benchmark which is increasingly used by hardware vendors, including ARM [17]. This benchmark evaluates CPU performance in terms of iterations per second. CoreMark code is compiled with *gcc* using maximum level of optimizations (`-O3`), the same as for compiling the MapReduce binaries. On the wimpy systems, we optimize the code for Cortex-A9 (`-mcpu=cortex-a9 -mtune=cortex-a9`) and Cortex-A15 (`-mcpu=cortex-a15 -mtune=cortex-a15`). Moreover, we enable fast NEON floating-point instructions (`-mfpu=neon`). ARM Cortex-A9 core is around two times slower than Cortex-A15 and four times slower than Intel i7 processor. However, A15's clock frequency is 65% higher, while i7's clock frequency is more than double compared to A9. Interestingly, A15 uses almost the same power per core as A9 to deliver twice the performance. This shows the significant improvements in energy efficiency of ARM processors. On the other hand, Intel processor uses five times more power compared to the wimpy processors. In terms of idle power, which is measured when the systems are running only the OS, A15 system is the most efficient, while the A9-only and i7-only systems consumes around two and eight time more power, respectively. We attribute the difference between the idle powers of the two wimpy system to the fact that A15 runs the OS on the very low-power companion core when idling. In summary, Cortex-A15 CPU achieves the best PPR, an observation which is consistent with the results for MapReduce workloads. Since the GPU is the key device in our evaluation, we measure peak performance in terms of FLOPS and data transfer bandwidth and latency between main (*host*) memory and GPU

[2] While Fig. 6 plots the normalized results with respect to the brawny system, absolute values are shown in Table A.6 in the Appendix.



(a) Bandwidth. (b) Latency.

Fig. 7. Host-device transfer bandwidth and latency.

(*device*) memory. We use MaxFlops to get arithmetic performance and BusSpeedDownload for host-device bandwidth, both from level zero benchmarks of Scalable HeterOgeneous Computing (SHOC) [18] benchmarking suite. But on A15 integrated system there is no point in measuring host-device bandwidth since data resides in the same shared memory. Thus, we additionally measure the latency using our custom benchmark that supports unified memory feature [19]. Benchmarking code is compiled with CUDA toolkit *nvcc* compiler with `-arch=sm_50` and `-arch=sm_32` flags for GTX 750 Ti and GK20A GPUs, respectively. For the Maxwell GTX 750 Ti GPU, the processing performance is the same on both A9 and i7 systems, peaking at more than 1500 GFLOPS when performing single-precision arithmetic. Interestingly, to deliver this performance, the i7+GPU draws more than two times the power of the A9+GPU, as shown in Fig. 6(b). We attribute this difference to at least three factors. First, by analyzing benchmark behavior, we discover that it highly utilizes one CPU core. From our CPU analysis, one i7 core draws around 25 W compared to around 2.5 W for one A9 core. Second, the PCI Express on A9 system consists of only four lanes, being more energy-efficient but having lower throughput. Although we are not able to measure PCI Express stand-alone energy usage, we believe it is much lower on the A9 system. Third, since A9 board is a low-power, embedded-class system, it uses more efficient circuitry, including power-regulators. In terms of idle power, the discrete Maxwell GPU consumes more than 10 W. Hence, the idle power of A9 doubles by adding this GPU. In contrast, A15's idle power is the same since the integrated GPU is not activated during idle periods. These results emphasize the energy efficiency of the A15 system.

For host-device transfers, the bandwidth is 40 times lower on the wimpy A9+GPU compared to i7+GPU, as shown in Fig. 7(a). This explains the poor performance of this configuration for MapReduce workloads that require significant transfers, such as GR. In terms of latency, A15 with integrated GPU is surprisingly not the best in all cases. While for larger transfers A15+GPU has a ten time lower latency, for small transfers of less than 128kB the i7+GPU shows lower latency. As expected, A9+GPU has the highest latency. This, together with the low bandwidth, seriously affects the ability of A9+GPU to process data-intensive applications.

It is a known fact that wimpy systems have smaller memories than brawny systems. Less well-known is the performance of these memories. We evaluate main memory bandwidth using *pmbw* tool [20]. Brawny system's memory bandwidth is 17 and 4 times higher compared to the A9 and A15 systems, respectively. Next, to get storage throughput we use Unix *dd* tool. SSD reads and writes have three and two times higher throughput respectively, on the brawny system compared to the wimpy A9 system. For the A15 system the difference in storage throughput compared to i7 is less than 20%. This difference is in part due to different SATA controller implementations. As modern operating systems buffer files in memory for subsequent reads, we also measure buffered read throughput. Brawny system uses its superior memory size and bandwidth to outperform the A9 and A15 systems at buffered reads by 20 and 6 times, respectively. In summary, wimpy systems have significantly lower performance, especially at memory level, which negatively impacts data analytics performance.

3.4. Bottleneck analysis

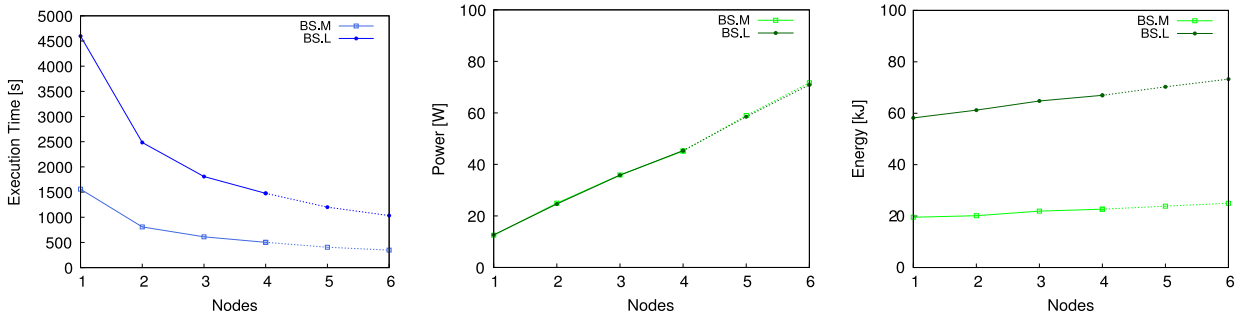
In this section, we use the time–energy, system profiling and system characterization results to discuss software and hardware improvements for more efficient MapReduce execution. Firstly, we analyze the scenarios in which the wimpy systems with powerful discrete GPUs, such as A9+GPU, become more energy-efficient. This is achievable by either (i) improving execution time or (ii) reducing GPU power, because energy is the product of execution time and average power,

$$E = T \cdot P. \tag{2}$$

Since our measurements show an average power usage ratio of two between A9+GPU and A9-only, the execution time of Hadoop–CUDA should exhibit a speedup of at least two. This could be obtained by highly compute-intensive workloads and by an improved Hadoop framework. On the other hand, given the maximum speedup of 1.2 shown by KM, the A9+GPU becomes more energy-efficient by reducing GPU power by 80%. This power reduction could be achieved by a wimpy platform

Table 4
Equivalence ratio between one brawny and multiple wimpy heterogeneous systems.

Workload	Systems	Equivalence ratio	Savings [%]	
			Time	Energy
BS.M	i7+GPU : A15+GPU	1 : 6	5	46
BS.L	i7+GPU : A15+GPU	1 : 6	−4	46
KM.M	i7+GPU : A15+GPU	1 : 3	−12	67
KM.L	i7+GPU : A15+GPU	1 : 3	0	68



(a) Time. (b) Power. (c) Energy.

Fig. 8. BS execution on clusters of A15+GPU.

with integrated GPU, such as Jetson TK1. Indeed, our measurements show that the GK20A GPU on Jetson TK1 consumes around 3 W when active, compared to around 25 W drawn by Maxwell GPU on the A9 system. Since the speedup on A15+GPU is also around 1.2, this configuration exhibits energy savings, as opposed to A9+GPU.

Secondly, we discuss the scenarios in which less compute-intensive workloads, such as GR, can benefit from GPU processing. On one hand, GR exhibits a large number of divergent control flow paths which results in high number of inactive CUDA threads, as shown in Section 3.2. For an input record that does not contain the searched string, a CUDA thread takes longer time to finish the processing. In contrast, for a record containing the string at the beginning, the CUDA thread finishes the processing very fast but must wait for the slowest thread in the warp. Since this is a workload characteristic, one solution is to improve control flow handling on GPUs. On the other hand, GR exhibits little computational work since it consists of scanning the input record, but requires large host-device data transfers. For example, on the brawny system, GR.L spends 30% of the time in data transfers. However, improving transfer bandwidth by three times results in 2% faster execution on CPU+GPU compared to CPU-only. This improvement is feasible as Nvidia already announced NVLink, a faster and more energy-efficient CPU–GPU path that can achieve a bandwidth of up to 200 GB/s [21].

3.5. Energy-efficient MapReduce execution

Analyzing the time–energy results across different systems, we observe that A9 configurations exhibit the highest execution times leading to energies that are similar or higher than those of brawny configurations. On the other hand, even if these brawny configurations exhibit the best execution time for all workloads, *intra-chip* heterogeneous wimpy system always consumes less energy. This result opens the alternative of using multiple A15+GPU nodes to perform the work of a single i7+GPU system with potentially less energy. We further analyze this opportunity for compute-intensive workloads with medium and large inputs on heterogeneous configurations. Small inputs are not representative for this analysis due to their small execution time dominated by Hadoop overheads. We measure the execution time and energy for clusters of up to four A15+GPU nodes and observe that for KM, three wimpy nodes achieve similar execution times as one i7+GPU, while saving almost 70% energy, as shown in Table 4. However, for BS more than four nodes are needed and, using the measured values, we estimate the time and energy using regression analysis. The execution time on n nodes,

$$T(n) = \frac{T(1)}{S(n)} \tag{3}$$

is determined by the speedup which should ideally be linear, $S(n) = n$. But in reality, the sequential fraction and the overheads of parallel and distributed execution lead to a sub-linear speedup. For BS, this sub-linear speedup determined an execution time that fits a power function, as shown in Fig. 8(a). Using the estimated execution times, we observe that six wimpy boards achieve the performance of one brawny system. Because energy is the product of execution time and average power, as in Eq. (2), on n nodes it becomes

$$E(n) = T(n) \cdot P(n) = \frac{T(1)}{S(n)} \cdot n \cdot P(1) = E(1) \cdot \frac{n}{S(n)} \tag{4}$$

assuming that the average power grows linearly with the number of nodes. This assumption is true for our workload as shown in Fig. 8(b). Since the speedup is sub-linear, the energy usage slowly increases with the number of nodes, as shown in Fig. 8(c). Even with this small increase, six wimpy nodes save 46% of the energy used by a single brawny system. These results advocate the usage of wimpy systems with integrated GPUs for compute-intensive data analytics.

4. Related work

The wide adoption of MapReduce and the advent of heterogeneous systems motivate the research in MapReduce frameworks supporting heterogeneity. However, the lack of works in analyzing the energy efficiency of MapReduce on heterogeneous systems with GPUs divides our related works discussion into (i) MapReduce on heterogeneous systems with GPUs that focus on time performance and (ii) energy efficiency of MapReduce.

4.1. MapReduce on heterogeneous systems with GPUs

Among the first MapReduce implementation for GPUs, Mars [13] provides the design and optimization techniques for a MapReduce framework on single-node systems with Nvidia GPUs. Mars achieves a maximum speedup of 16 compared to Phoenix [22], a CPU-only C++ implementation of MapReduce. However, Mars has at least two limitations. First, it is only a shared-memory implementation and does not support clusters of nodes with GPUs. Second, it does not consider energy usage. Nevertheless, Mars represents a major breakthrough followed by a series of works proposing incremental improvements [23–26]. Among these, Chen et al. propose a MapReduce framework targeting AMD Fusion architecture which integrates the CPU and GPU on the same chip [25]. To utilize both the CPU and GPU, the authors use two different approaches: (i) a map-dividing approach in which both devices run map and reduce tasks and (ii) a pipelining approach in which one device executes map tasks and the other runs the reduce tasks. Using scheduling and tuning techniques, their CPU–GPU framework achieves a maximum speedup of 2 compared to the best of CPU- and GPU-only approach.

GPMR is one of the first stand-alone MapReduce frameworks for clusters with GPUs [27]. In addition to the framework design and implementation, the authors introduce a series of optimization techniques for implementing MapReduce on GPUs, such as partial reductions and accumulation of ⟨*key*, *value*⟩ pairs on GPU. GPMR is evaluated on a cluster with 64 NVIDIA Tesla GPUs but shows poor scalability when running on more than eight GPUs. Moreover, this framework does not implement fault-tolerance and distributed file system (DFS) support. Glasswing [28] is a MapReduce framework for cluster nodes with GPUs that implements a pipeline of five stages to overlap computation with communication in map and reduce. Glasswing provides a MapReduce-style OpenCL API to exploit the intra-node heterogeneity. However, Glasswing is impractical to use in real-world applications because it has limited data management and fault-tolerance.

Since Hadoop [2] is the most popular MapReduce implementation, some research projects explore its usage on clusters of heterogeneous systems with GPUs. Among the first projects to explore this, MITHRA [10] uses Hadoop and CUDA to improve the execution of embarrassingly parallel applications on clusters of nodes with GPUs. MITHRA's implementation of BlackScholes, the only application evaluated, achieves better execution time on four nodes with GPUs compared to a cluster of 62 CPU-only nodes. Shirahata et al. propose a hybrid CPU–GPU map phase scheduling for Hadoop [12]. They evaluate the Kmeans clustering algorithm by implementing its map phase in C++ and CUDA and integrate it on Hadoop through Pipes mechanism. To efficiently schedule map tasks on both CPU and GPU, they propose to solve a minimization problem for task execution. The input parameters for this problem, such as map task time and GPU acceleration, are obtained after profiling the first wave of map execution. Using the proposed scheduling technique, the maximum speedup over Hadoop on a 64-node cluster is 1.93. HadoopCL [29] combines Hadoop and OpenCL and automatically generates OpenCL code from Java code using a tool called APARAPI. But this translation tool limits the capabilities of Java code since only primitive types and a few library methods can be used. Although energy efficiency is a main motivation for this work, the authors state that no energy analysis is presented due to the lack of infrastructure to measure power and energy.

Our work leverages on Hadoop's distributed execution and fault tolerance at cluster level, while focusing on exploiting the intra-node heterogeneity of systems with GPUs. Our *lazy processing* technique is workload-independent and eases developer's task by proposing to distribute entire ⟨*key*, *value*⟩ pairs to GPU cores instead of splitting the processing. Our technique requires no modifications to the underlying Hadoop framework and, thus, can be more easily adopted in real-world deployments. To the best of our knowledge, we are the first to provide an energy usage analysis of MapReduce on heterogeneous systems with GPUs.

4.2. Energy efficiency of MapReduce

As the previous works do not address the energy efficiency of MapReduce on heterogeneous systems with GPUs, we discuss the energy usage of MapReduce on homogeneous systems. There are two techniques for saving energy during low-utilization periods. One of them, called Covering Set (CS) [30], shuts-down all the nodes in the cluster during low utilization intervals, except a small set (the Covering Set) of nodes which store at least one replica of each HDFS block. The other, All-In Strategy (AIS) [31], is based on the claim that it is more energy-efficient to use all the nodes and, thus, finish MapReduce jobs faster and shut-down all nodes afterwards. Berkeley Energy Efficient MapReduce (BEEMR) [32], proposes

to split a MapReduce cluster into interactive and batch zones. The nodes in batch zone are kept in a low-power state when inactive. This technique is based on the insights from MapReduce with Interactive Analysis (MIA) workloads. For this kind of workloads, interactive MapReduce jobs tend to access only a small part of data. Hence, an interactive cluster zone can be obtained by identifying these interactive jobs and their required input data. The rest of the jobs are executed on the batch zone at defined time intervals. Using both simulation and validation on Amazon EC2, BEEMR reports energy savings of up to 50%. Feller et al. study the performance and power consumption of Hadoop on clusters with collocated and separated data and compute nodes [33]. They highlight two unsurprising findings: (i) the PPR of collocated data and compute nodes is better compared to a separated deployment and (ii) power is dependent on MapReduce phases. While our approach and analysis target intra-node and intra-chip heterogeneity, these related works complement our work and can be applied at cluster level.

As an exception, Loghin et al. analyze the energy usage and total cost of ownership for MapReduce applications on intra-chip heterogeneous systems represented by ARM big.LITTLE [34]. While these systems integrate CPU cores with different PPR, our intra-chip heterogeneous systems integrate CPU cores and GPU cores with both different PPR and different architecture. Similar to us, the authors show that is non-trivial to decide which system is better for a given application and that, even if wimpy systems use less power, execution inefficiencies lead to high energy usage.

5. Conclusion

In this paper, we analyze the time–energy performance of MapReduce on heterogeneous systems with GPUs, in comparison with homogeneous CPU-only systems. Moreover, with the performance improvements of wimpy systems traditionally used in mobile devices, we investigate their performance on processing data analytics compared to brawny server systems. We select three systems representing both *intra-node* and *intra-chip* heterogeneity: (i) an Intel i7 system hosting a discrete 640-core Nvidia GPU of the latest Maxwell generation, representing *intra-node* heterogeneous brawny systems, (ii) a quad-core ARM Cortex-A9 with the same Maxwell GPU representing *intra-node* heterogeneous wimpy systems, and (iii) a quad-core ARM Cortex-A15 integrated with 192 Nvidia Kepler GPU cores representing *intra-chip* heterogeneous wimpy systems. We evaluate the time and energy performance of these systems using three MapReduce applications with diverse resource demands. To execute MapReduce on heterogeneous systems with GPUs, we introduce a novel *lazy processing* technique which simplifies application development and requires no modifications to the underlying Hadoop framework. Our analysis is based on MapReduce time–energy measurements, workload execution profiling and system characterization at the CPU, GPU, memory and storage levels.

For compute-intensive workloads such as BlackScholes, the brawny heterogeneous achieves speedups of up to 2.3 and reduces the energy usage by almost half compared to the brawny homogeneous system. As expected, for applications such as Grep where data transfers dominate the execution time, heterogeneous systems exhibit worse time–energy performance compared to homogeneous systems. For example, the heterogeneous wimpy A9 with discrete GPU consumes 14 times the energy of the homogeneous A9 system due to very low host-device transfer bandwidth and high power overhead of the discrete GPU. Moreover, the lower performance of wimpy systems on data analytics is in part due to the small main memory size. While brawny systems have large memories to accommodate Hadoop's intermediate data, our profiling shows that on both wimpy systems data is spilled to disk leading to 80% more storage transfers compared to the brawny system. Among heterogeneous systems, the wimpy with discrete GPU exhibits the worst time–energy performance. But the wimpy with integrated GPU uses the lowest energy across all workloads due to more energy-efficient CPU and GPU, and better balanced system resources. To account for the execution time difference, we establish an equivalence ratio between a single brawny heterogeneous node and multiple wimpy heterogeneous nodes. Based on this equivalence, the wimpy nodes not only achieve similar execution times compared to a single brawny node, but also exhibit energy savings of up to two-thirds. This result advocates the potential usage of wimpy systems with integrated GPUs for Big Data analytics.

Acknowledgments

We are grateful to Nvidia for providing us with four Jetson TK1 boards. This work was in part supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Competitive Research Programme (CRP Award No. NRFCRP8-2011-08).

Appendix. Additional data

In this appendix, we present the absolute values of system characterization at the CPU, GPU, memory and storage levels, as described in Section 3.3. Additionally, we argue that the Intel i7 CPU usually employed in desktop PCs has higher PPR compared to an Intel Xeon typically used in server systems. These two processors of the same generation have similar specifications, except that i7 implements hyper-threading and is clocked at two times the frequency of Xeon, as shown in Table A.5. Moreover, i7 exhibits higher performance, as shown in Table A.6. Firstly, it has a better PPR compared to the Xeon since at the same average power of around 50 W, it delivers almost twice the performance per core. This high performance is due to higher clock frequency. In addition, the lower idle power of i7 system compared to Xeon system further advocates

Table A.5
Systems specification.

Specifications		i7	Xeon
CPU	Type	Intel Corei7	Intel Xeon E5-2603
	ISA	x86-64	x86-64
	Cores	4 (8 threads)	4
	Frequency [GHz]	1.60–3.40	1.20–1.80
	L1 cache (per core)	32 kB	32 kB
	L2 cache	1 MB	1 MB
	L3 cache	8 MB	10 MB
Memory		16 GB DDR3	8 GB DDR3
Storage		512 GB SSD	512 GB SSD

Table A.6
Systems characterization.

Subsystem	Tool	Metric	i7	Xeon	A9	A15
CPU	CoreMark	Performance per core [iterations/s]	17237	9456	3952	8155
		Average system power [W][a]	51.7	50.6	10.1	5.9
		PPR [iterations/J]	333.4	186.9	391.3	1382.2
		Idle system power [W]	25.8	35.0	7.7	3.2
GPU	SHOC	Performance [GFLOPS]	1514	N/A	1512	209
		Average system power [W]	105.2	N/A	44.0	6.1
		Idle system power [W]	40.6	N/A	19.2	3.2
Memory	pmbw	Bandwidth [GB/s]	17.6	12.2	0.9	4.3
Storage	dd	Write throughput [MB/s]	198	215	90	161
		Read throughput [MB/s]	284	479	85	275
		Buffered read throughput [GB/s]	10.0	4.8	0.4	1.6

[a] Power is measured on the systems without GPU and with the benchmark running on a single CPU core.

the energy efficiency of this system. Secondly, the 44% higher memory bandwidth gives i7 an advantage at processing data analytics. In summary, we use the i7 brawny system with better performance to assess the energy efficiency limits of wimpy systems.

References

[1] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Proc. of 6th Conference on Symposium on Opearting Systems Design & Implementation, pp. 10–10.
[2] Apache, Hadoop, 2014. http://hadoop.apache.org/.
[3] K.-H. Lee, Y.-J. Lee, H. Choi, Y.D. Chung, B. Moon, Parallel data processing with MapReduce: A survey, SIGMOD Rec. 40 (2012) 11–20.
[4] L. Ramapantulu, B.M. Tudor, D. Loghin, T. Vu, Y.M. Teo, Modeling the energy efficiency of heterogeneous clusters, in: Proc. of 43rd International Conference on Parallel Processing.
[5] V. Gupta, K. Schwan, Brawny vs. Wimpy: Evaluation and analysis of modern workloads on heterogeneous processors, in: Proc. of 27th International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum, pp. 74–83.
[6] A. Cunningham, From smartphone to server room: Nvidia's "Kayla" shows the future of Tegra, 2013. http://www.webcitation.org/6VcpwYsD5.
[7] NVIDIA unveils first mobile supercomputer for embedded systems, 2014. http://www.webcitation.org/6VdkUISQn.
[8] N. Hemsoth, Breaking: Detailed results from today's Top 500 fastest supercomputers list, 2014. http://www.webcitation.org/6W25X8tj4.
[9] M. Harris, 5 things you should know about the new Maxwell GPU architecture, 2014. http://www.webcitation.org/6VcZH7xTv.
[10] R. Farivar, A. Verma, E. Chan, R. Campbell, MITHRA: Multiple data independent tasks on a heterogeneous resource architecture, in: Proc. of 2009 IEEE International Conference on Cluster Computing and Workshops, pp. 1–10.
[11] C. Bienia, S. Kumar, J.P. Singh, K. Li, The PARSEC benchmark suite: characterization and architectural implications, in: Proc. of 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 72–81.
[12] K. Shirahata, H. Sato, S. Matsuoka, Hybrid map task scheduling for GPU-based heterogeneous clusters, in: Proc. of 2nd International Conference on Cloud Computing Technology and Science, pp. 733–740.
[13] B. He, W. Fang, Q. Luo, N.K. Govindaraju, T. Wang, Mars: A MapReduce framework on graphics processors, in: Proc. of 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 260–269.
[14] D. Jiang, B.C. Ooi, L. Shi, S. Wu, The performance of MapReduce: An in-depth study, Proc. VLDB Endow. 3 (2010) 472–483.
[15] A. Vermaa, L. Cherkasovab, R.H. Campbellc, Profiling and evaluating hardware choices for MapReduce environments: An application-aware approach, Perform. Eval. 79 (2014) 328–344.
[16] GNU, GCC ARM options, 2015. http://www.webcitation.org/6VazB3S7x.
[17] ARM, ARM announces support for EEMBC CoreMark benchmark, 2009. http://www.webcitation.org/6RPwNECop.
[18] A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter, The scalable heterogeneous computing (SHOC) benchmark suite, in: Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, New York, NY, USA, pp. 63–74.
[19] M. Harris, Unified memory in CUDA 6, 2013. http://www.webcitation.org/6WrmVnwyE.
[20] T. Bingmann, Parallel memory bandwidth benchmark/measurement, 2013.
[21] D. Foley, NVLink, pascal and stacked memory: Feeding the appetite for big data, 2014. http://www.webcitation.org/6XL0t4iVi.
[22] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, Evaluating MapReduce for Multi-core and Multiprocessor Systems, in: Proc. of 13th International Symposium on High Performance Computer Architecture, pp. 13–24.
[23] C. Hong, D. Chen, W. Chen, W. Zheng, H. Lin, MapCG: Writing parallel program portable between CPU and GPU, in: Proc. of 19th International Conference on Parallel Architectures and Compilation Techniques, pp. 217–226.

[24] F. Ji, X. Ma, Using shared memory to accelerate MapReduce on graphics processing units, in: Proc. of 25th IEEE International Parallel Distributed Processing Symposium, pp. 805–816.
[25] L. Chen, X. Huo, G. Agrawal, Accelerating MapReduce on a coupled CPU–GPU architecture, in: Proc. of 2012 International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 25:1–25:11.
[26] L. Chen, G. Agrawal, Optimizing MapReduce for GPUs with effective shared memory usage, in: Proc. of 21st International Symposium on High-Performance Parallel and Distributed Computing, pp. 199–210.
[27] J.A. Stuart, J.D. Owens, Multi-GPU MapReduce on GPU clusters, in: Proc. of 25th IEEE International Parallel and Distributed Processing Symposium, pp. 1068–1079.
[28] I. El-Helw, R. Hofman, H.E. Bal, Scaling MapReduce vertically and horizontally, in: Proc. of 2014 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 525–535.
[29] M. Grossman, M. Breternitz, V. Sarkar, HadoopCL: MapReduce on distributed heterogeneous platforms through seamless integration of hadoop and OpenCL, in: Proc. of 27th International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum, pp. 1918–1927.
[30] J. Leverich, C. Kozyrakis, On the energy (in)efficiency of hadoop clusters, SIGOPS Oper. Syst. Rev. 44 (2010) 61–65.
[31] W. Lang, J.M. Patel, Energy management for MapReduce clusters, Proc. VLDB Endow. 3 (2010) 129–139.
[32] Y. Chen, S. Alspaugh, D. Borthakur, R. Katz, Energy efficiency for large-scale MapReduce workloads with significant interactive analysis, in: Proc. of 7th ACM European Conference on Computer Systems, pp. 43–56.
[33] E. Feller, L. Ramakrishnan, C. Morin, On the performance and energy efficiency of Hadoop deployment models, in: Proc. of 2013 IEEE International Conference on Big Data, pp. 131–136.
[34] D. Loghin, B.M. Tudor, H. Zhang, B.C. Ooi, Y.M. Teo, A performance study of big data on small nodes, Proc. VLDB Endow. 8 (2014) 762–773.

Dumitrel Loghin is a Ph.D. student at School of Computing, National University of Singapore. He received the Master degree in Advanced Computer Architectures from University Politehnica of Bucharest, Romania, and the Bachelor degree in Computer Science from the same university. His research interests include parallel and distributed computing and the performance of computer systems.

Lavanya Ramapantulu is a Ph.D. student at School of Computing, National University of Singapore. She received the Master degree in Microelectronics from Birla Institute of Technology and Science, Pilani, India and the Bachelor degree in Electronics and Communication Engineering from National Institute of Technology, Karnataka, India. Her research interests are in the area of performance analysis, parallel and distributed computing, analytical modeling and parallel computer architecture.

Oana Barbu is a research assistant at School of Computing, National University of Singapore. She received the Master degree in Computer Network Security, and Bachelor degree in Computer Science from University Politehnica of Bucharest, Romania. Her research interests include parallel and distributed computing and performance analysis.

Yong Meng Teo received the B.Tech (1st Class Hons) in Computer Science from the University of Bradford, UK, and the MS and Ph.D. in Computer Science from the University of Manchester, UK. He is an Associate Professor of Computer Science and an Affiliate Professor (Business Analytics Centre) at the National University of Singapore. His research interests are in parallel and distributed computing, performance analysis and systems modeling and simulation.