2015 IEEE First International Conference on Big Data Computing Service and Applications

Power Efficient MapReduce Workload Acceleration using Integrated-GPU

SungYe Kim, Jeremy Bottleson, Jingyi Jin, Preeti Bindu, Snehal C. Sakhare, Joseph S. Spisak
Intel Corporation, Folsom, CA 95630
{sungye.kim, jeremy.bottleson, jingyi.jin, preeti.bindu, snehal.c.sakhare, joseph.s.spisak}@intel.com

***Abstract*—With the pervasiveness of MapReduce - one of the most prominent programming models for data parallelism in Apache Hadoop -, many researchers and developers have spent tremendous effort attempting to boost the computational speed and energy efficiency of MapReduce-based big data processing. However, the scalable and fault-tolerant nature of MapReduce introduces additional costs in disk IO and data transfer, caused by streaming intermediate outputs to disk. In light of these issues, many interesting research projects have been initiated with the goal of improving the compute speed and power efficiency of compute-intensive cloud computing workloads; several with the addition of discrete GPUs. In this work, we present a modified MapReduce approach focused on the iterative clustering algorithms in the Apache Mahout machine learning library that leverage the acceleration potential of the Intel integrated GPU in a multi-node cluster environment. The accelerated framework shows varying levels of speed-up (≈45x for Map tasks-only, ≈4.37x for the entire K-means clustering) as evaluated using the HiBench benchmark suite. Based on various experiments and in-depth analysis, we find that utilizing the integrated GPU via OpenCL offers significant performance and power efficiency gains over the original CPU based approach. Further analysis is also done to understand the correlations between compute, IO and power efficiency. As such, our results show that embracing the integrated GPU in the Hadoop MapReduce framework represents a promising advance in adding cost and energy efficient compute parallelism to a data parallel multi-node environment.**

I. Introduction

With the rise of ubiquitous high speed internet, enormous amounts of data are generated daily to store profiles and to log events. For instance, the amount of global Internet data traffic for mobile devices was 1.5 exabytes per month in 2013 [1]. To turn that large amount of data into useful insights and human manageable information, tremendous effort has been spent by both academia and industry. The study of processing or analyzing data which is not suited or cannot be handled in a single-node environment has been generally referred to as Big Data Processing. Big data is typically stored in large data centers using distributed cluster based systems. One commonly used system, Apache Hadoop [2], composed mainly by HDFS (Hadoop Distributed File System) and the MapReduce compute model, is a framework for reliably and easily storing and processing large amounts of data across a multi-node cluster. From a system design point of view, Hadoop consists of a NameNode, which plays the role of master, to manage resources and assign Map or Reduce tasks to the registered DataNodes, or child nodes. The Map and Reduce tasks are performed independently on the DataNodes by reading parts of input, and generating output when the processing is done. Because of this repetitive and independent nature of the Map and Reduce tasks, and the fact that they usually constitute the largest chunk of processing time, several attempts have been targeted to accelerate those tasks [3]–[6].

The incredible computing capabilities of GPUs as well as low power consumption motivated us to leverage the newest integrated GPUs to accelerate MapReduce workloads, as data centers typically rely on CPU-compute to process workloads. Due to the integrated and high performance per watt nature of current integrated GPUs, we believe that when utilized for compute-intensive code blocks, it can present equivalent or potentially better performance compared to CPU, yet with lower total power consumption. Our goal with this work is to demonstrate the applicability of adapting already existing MapReduce algorithms to leverage the integrated GPU, and to verify performance and power gains over the pure CPU implementation. To accomplish this goal, we used Intel integrated GPUs, and started with K-means clustering in Apache Mahout [7], a commonly used machine learning library for Hadoop, to support GPU execution using OpenCL. We then made further optimizations to leverage some of the advantages of the integrated GPU such as the shared CPU/GPU memory, as well as the efficient caching hierarchy. During our optimization effort to boost performance, we discovered that the workload bottleneck shifted from compute side to IO side due to the IO intensive nature of MapReduce. In the next sections, we present the details of our optimization process, as well as the results, analysis and discussions.

II. Related Work

In big data computing, several techniques using Apache Hadoop focus on reducing various overheads within the framework and/or accelerating the MapReduce programming model. Some previous approaches include optimized scheduling mechanisms [8]–[11], and GPU powered frameworks [4], [12]–[14], with efficient GPU memory usages [15]–[19].

In the field of optimizing MapReduce task scheduling schemes, Chen et al. [10] describe MapReduce task scheduling and pipelining schemes to balance loads on CPU and GPU while keeping low scheduling costs. To achieve this, they use a runtime tuning method and perform experiments on AMD Fusion APU (Accelerated Processing Unit), integrating CPU and GPU. However, the authors leave extending their framework to a multi-node cluster as future work. Gu et al. [11] present an optimized Hadoop framework, called SHadoop, which improves MapReduce job scheduling and task execution mechanisms by focusing on an instance messaging communication between JobTracker and TaskTrackers to minimize task

978-1-4799-8128-1/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataService.2015.12

execution time. This work has also been integrated into the Intel Distributed Hadoop.

A variety of efforts have been made to investigate building GPU powered custom Hadoop frameworks to accelerate MapReduce as well as provide a convenient programming interface for the GPU [4], [12]–[14], [20]. He et al. [12] propose a MapReduce framework based on NVidia CUDA, called Mars, which achieves 1.5-16x performance speed-up over a CPU-based MapReduce framework, Phoenix [21]. In the Mars framework, the authors propose a two-step scheme for the Map and Reduce stages to calculate the precise amount of memory on the device to allocate for arrays for storing outputs from the Map and Reduce stages. Although their approach aims for massive thread parallelism on GPU, their framework essentially doubles the map and reduce tasks executed. MapCG by Hong et al. [20] describes a MapReduce based programming model, which is portable between CPU and GPU, and implements a dynamic memory allocator and a hash table on GPU by using atomic instructions, incurring performance penalties.

Besides harnessing GPUs in Hadoop frameworks, there has been research to accelerate Java programs with heterogeneous devices. AMD released Aparapi [22], which is an open source API generating OpenCL kernels from Java bytecode allowing workloads in Java to be executed on OpenCL capable devices like CPU, GPU and APU. However, Aparapi only supports some of Java features such as simple loops, primitive data types, and one-dimensional arrays rather than references to arbitrary objects. Hayashi et al. [23] propose an approach called HJ-OpenCL creating OpenCL kernels from programs in HJ (Habanero-Java) language by integrating Aparapi as well as enabling static translation from Java bytecode to OpenCL. Likewise converting Java bytecode to OpenCL kernels, Calvert's work [24] creates a compiler to allow Java bytecode to be executed on NVidia CUDA framework. Rootbeer [25] is also an open source library converting Java code to CUDA programs. In contrast to Aparapi, Rootbeer serializes the complete relevant Java objects to a native format compatible with CUDA. In the field of distributed systems, Grossman et al. [26] present HadoopCL that is an extended Hadoop system utilizing Aparapi to convert Java code to OpenCL kernels in their system, combining Hadoop and OpenCL to accelerate Hadoop workloads in a heterogeneous environment.

Researchers have also focused on utilizing fast on-chip shared memory to enhance the performance of MapReduce applications. Ji and Ma [16] explore performance variation under different GPU memory usages (global vs. shared memory), and reveal that GPU shared memory provides a promising performance gain of 2.85x in the Map phase. This work also removes the cost of the two pass execution in the Mars [12] framework by adopting atomic global operations. Although they alleviate global memory access with shared memory, there exists potential synchronization overhead in using atomic operations. A reduction based method proposed by Chen and Agrawal [18] reduces the overhead in copying and synchronizing data on the device memory when executing applications. Their approach inserts each intermediate output to the reduction object at the end of the Map phase in order to perform reduction on the shared memory. However, their work is only applicable to commutative and associate map and reduce operations due to the nature of reduction approach.

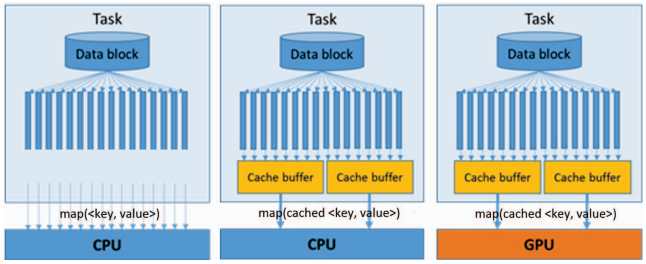


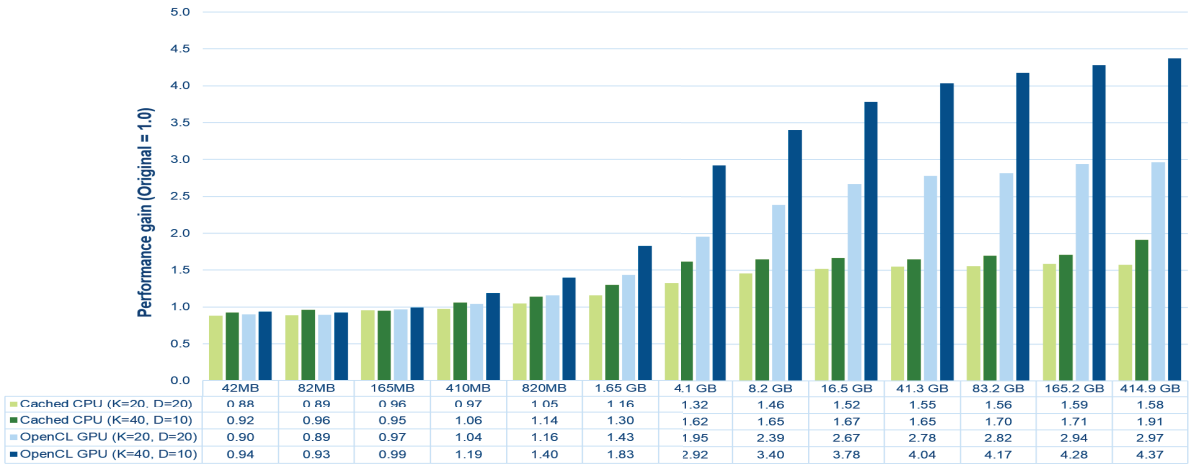
Fig. 1. Illustration of algorithmic change to the Map task. (Left) Original design of a Map task. Data block is split into key/value pair and sent for processing by calling map function. (Middle) Added cache buffers to alleviate intensive map call to CPU. (Right) Map calls are replaced with OpenCL kernels, thus the compute is offloaded to GPU, when that resource is available.

Unlike discrete graphics processors which have been extensively used in studies to accelerate MapReduce, there has been little research [5], [6] in investigating the potential of integrated GPUs for the MapReduce paradigm in distributed environments. Discrete vs. integrated GPUs have their own set of tradeoffs. Discrete GPUs provide more computing capability but at a higher price and with high power consumption, and data must be transferred between host and GPU Memory over PCIe creating more IO overhead and a potential bottleneck. Integrated GPUs can provide significant benefits in that CPU and GPU share the same physical memory and can pass data between themselves with 0 copies and no need to use the PCIe bus. Kim et al. [5] evaluate the performance of discrete GPUs and integrated GPUs with memory intensive workloads by quantifying interference between CPU and GPU, based on memory transfer rate between CPU and GPU as well as memory access ratio on CPU and GPU when workloads run in an APU. Their work shows that memory intensive workloads are more affected by the CPU-GPU interference because they have low computation per memory access. Furthermore, the authors indicate that better performance in time does not always imply better energy efficiency. Recently, Ching et al. [6] investigated using integrated GPUs for database workloads with multiple query and primitive operations. Based on their experiments with micro-benchmarks, the authors conclude that integrated GPUs have better marginal performance return in terms of capital and operation cost, but there still remains further exploration with multi-node environments and more complex workloads as future work.

Finally, it is important to note that our work has a similar goal to that of Ching et al. [6] in that we both conclude on the benefits of using integrated GPUs in the form of Intel®HD Graphics. However, our work is more focused on real world workloads for big data processing in a multi-node cluster environment as well as performs various experiments to leverage optimization possibilities as described in following sections.

III. Integrated GPU-accelerated MapReduce

Our approach, using integrated GPUs to accelerate MapReduce workloads, modifies algorithmically Map or Reduce tasks to support GPU acceleration. When analyzing some common workloads, it was identified that usually Map tasks consisted



	42MB	82MB	165MB	410MB	820MB	1.65 GB	4.1 GB	8.2 GB	16.5 GB	41.3 GB	83.2 GB	165.2 GB	414.9 GB
Cached CPU (K=20, D=20)	0.88	0.89	0.96	0.97	1.05	1.16	1.32	1.46	1.52	1.55	1.56	1.59	1.58
Cached CPU (K=40, D=10)	0.92	0.96	0.95	1.06	1.14	1.30	1.62	1.65	1.67	1.65	1.70	1.71	1.91
OpenCL GPU (K=20, D=20)	0.90	0.89	0.97	1.04	1.16	1.43	1.95	2.39	2.67	2.78	2.82	2.94	2.97
OpenCL GPU (K=40, D=10)	0.94	0.93	0.99	1.19	1.40	1.83	2.92	3.40	3.78	4.04	4.17	4.28	4.37

Fig. 2. Performance gains in K-means Job processing time of our cached CPU and OpenCL GPU versions with two data configurations (higher is better, original K-means Job processing time = 1.0).

of the largest portion of processing time, while Reduce tasks perform more IO processing, like writing results into HDFS rather than compute. In our case study of K-means, we focus on modifying the Map task, and our two-step algorithmic change to a Map task or Mapper is illustrated in Fig. 1.

A. OpenCL-based Mapper

In a Map task, a map function is executed on each key/value pair requiring a large number of map calls to process all data on each DataNode, as illustrated in Fig. 1 (left). As such, some overhead from per-map IO and intercommunication between JobTracker and TaskTracker is inevitable. To reduce the per-map call overhead as well as prepare batched data for OpenCL kernels, we utilize a data caching strategy (Fig. 1 (middle)). A fixed amount of key/value pairs are cached into a cache buffer and the map function then processes the cached key/value pairs at once. At last, to support GPU acceleration in the Mapper, OpenCL kernels are created to replace the old map call, which leads the compute being offloaded to GPU (Fig. 1 (right)), when it is available.

In our framework the cache buffer size is determined based upon experimental analysis of the workload and GPU hardware in offline, and then fixed throughout the execution of the map task. Map tasks which do not contain enough key value pairs to fill the cache are padded. This avoids costly reallocation of GPU buffers during map task execution.

In the Hadoop framework, since the Mappers run inside their own JVM (Java Virtual Machine) and they do not share any resources, we need to setup an OpenCL device per Mapper. In the setup method of each Mapper, we initialize OpenCL by getting the OpenCL platform and GPU device through JOCL Java binding [27]. Furthermore, since we use a fixed size cache buffer, all required OpenCL buffers are allocated at initialization. Finally, we created a cached CPU version without OpenCL kernels, which provides a useful apparatus to analyze performance differences between algorithmic changes from the original CPU version with the introduction of cache, and the GPU acceleration.

TABLE I. CLUSTER CONFIGURATION.

	4-node cluster	*1-node cluster*
Nodes	1 name node, 3 data nodes	1 name/data node
GPU	Intel HD Graphics 4600	Intel HD Graphics 4600, NVidia Titan Z
OpenCL	Intel OpenCL 1.2	Intel OpenCL 1.2, NVidia OpenCL 1.1
CPU	8 logical-core Intel® Core™ i7-4770K @ 3.5GHz	8 logical-core Intel® Xeon® E3-1275 v3 @ 3.5GHz
RAM	32GB (on each node)	
OS	CentOS 6.5 Kernel 3.14.5	
Hadoop	Cloudera Hadoop 2.3.0-CDH 5.0.2 MapReduce2	
Java	JDK 1.7.0.45	
Java-OpenCL	JOCL 0.1.9	

B. Optimization

Creating the GPU accelerated execution paths can be roughly split into a few different categories, host-side optimizations, kernel-side optimizations and algorithmic optimizations.

On the host side, in order to create an optimal solution, one must try to reduce the amount of unnecessary data copying in order to efficiently transfer data to and from the GPU. It is also imperative to pass data to the GPU in large enough chunks so that the GPUs' data parallelism can be exploited. To accomplish this, we use the caching mechanism mentioned in Sec. III to create a buffer containing a set of key/value pairs effectively saturating the GPUs' compute. On integrated graphics, we also take advantage of the fact that GPU and CPU share the same physical memory. Therefore, we optimize our host side performance by allocating OpenCL buffers once on the host and use them directly in OpenCL kernels rather than making copies. In addition to data copying overhead, there also exists kernel call overhead in each OpenCL kernel. In an attempt to reduce the call overhead, we minimize the number of kernel execution calls by sending more data on each GPU kernel call, and/or combining kernels whenever possible.

Kernel optimizations can be made by taking advantage of hardware specific features. For instance, a commonly used approach is the use of shared local memory which is a small amount of memory shared by work items in the same workgroup on the GPU, thereby it provides fast access from all work items in a workgroup. On Intel® HD Graphics hardware, shared local memory is equivalent in speed to the

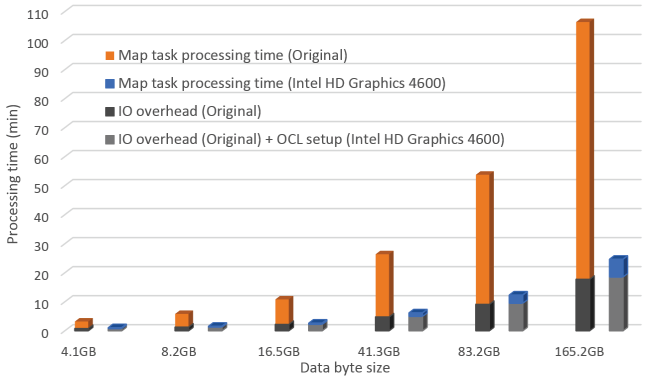
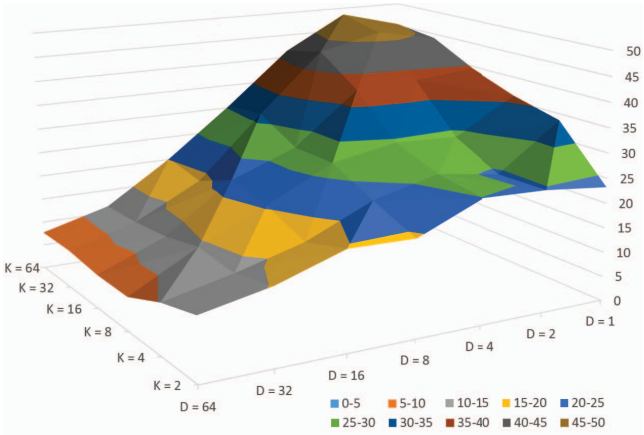


Fig. 3. Comparison of total processing time for original CPU and OpenCL GPU split into IO overhead and map-only processing time (4-node cluster).



	D = 1	D = 2	D = 4	D = 8	D = 16	D = 32	D = 64
K = 2	23	25	25	19	20	15	13
K = 4	34	24	26	23	17	13	10
K = 8	36	29	29	27	20	14	7
K = 16	40	38	38	27	22	12	7
K = 32	44	47	43	31	19	12	8
K = 64	45	49	41	30	20	11	8

Fig. 4. Performance gains in K-means map tasks time only of our OpenCL GPU version with various data configurations (higher is better, original K-means map tasks time = 1.0). The data size is constant: ≈17GB.

L3 cache, making it much faster than global memory. Kernels with several work items accessing the same address in global memory can benefit greatly by first reading the data into shared local memory and then accessing it multiple times. In our optimized framework, we implement different kernel versions for enabling the use of shared local memory. Non-shared local memory kernels are used as a fallback in cases where the data per workgroup exceeds the device memory per workgroup, while shared local memory kernels are used in default cases.

Algorithmic optimizations also play an important role in creating an optimal GPU execution path. GPUs are SIMD (Single Instruction Multiple Data) devices and their performance can be considerably impaired when including too many branches in the kernel code. It thus is a critical design consideration to move branches to the host side and out of the kernel, if possible. Another design consideration comes from the fact that memory cannot be dynamically allocated inside a kernel so that any outputs need to have pre-allocated space on the host side. In our design, we separate the different clustering algorithm execution paths into separate GPU accelerated pathways. While this has the drawback of increasing overall code size, it presents the advantage of allowing us to create OpenCL kernels which perform the same function as multiple separate modules in the original design. It also allows us to create kernels with a minimal amount of branching, optimize the flow of data through the kernel pipeline to minimize data copying, and finally optimize the design to take advantage of the fact we will be processing more than 1 sample at a time. As such, we consider these changes necessary for optimal GPU execution.

IV. Experimental Evaluation

A. Apparatus

Cluster Configuration: We performed experiments on a 4-node cluster as well as a single-node cluster, with hardware/software configurations as described in Table I. To have a better understanding of performance benefits per node, we setup a homogeneous multi-node cluster. For comparison on discrete vs. integrated GPU, we built two single-node clusters with identical hardware configurations, but one with the Intel HD Graphics 4600 and the other with the addition of NVidia Titan Z discrete card.

Benchmark suites and Workloads: To make the evaluation reproducible by other studies, we use HiBench [28] 2.2 as the benchmark standard. HiBench suite contains sets of synthetic and real-world workloads to test the performance of applications over Hadoop. Other benchmark suites, such as BigBench [29], were also considered. However, the overhead of running Hive queries to prepare the data for the workloads is very large in BigBench, making the benchmark process more difficult on our 4-node cluster. For instance, BigBench Query 26 runs over 20 hours to prepare Hive tables by generating raw text data (e.g., 10TB), transforming them into ORC (Optimized Row Columnar) format, and executing Hive queries to insert data into Hive tables. From such data preparation stages, relatively small sequence data files are generated for a data analysis stage (e.g., 300MB). A possible course of action was to increase the size of sequence data files by generating larger raw data, however this would require additional time and effort for data preparation. Hence, we chose to focus on accelerating Map tasks, assuming that big data already exists in a form of either database or warehouse in real world cases. We finally opted for HiBench, a common benchmark suite used in industry, as it provides easy and direct control over the data generation in various data sizes.

Some research greatly characterizes common data center workloads, such as service workloads and data analysis workloads [30]–[32]. Jia et al. [31] investigate various factors such as IPC (Low instruction Per Cycle) and DWS (Disk Writes per Second) on multiple workloads, and show that data analysis workloads tend to have higher IPC than service workloads. Among eleven data analysis workloads they investigate (i.e., Naive Bayes, Support Vector Machine, Grep, WordCount, K-means, Fuzzy K-means, PageRank, Sort, Hive-bench, Item Based Collaborative Filtering, Hidden Markov Model), Naive Bayes and K-means show considerably low DWS, which implies low IO-bound, but exceptionally Naive Bayes has the

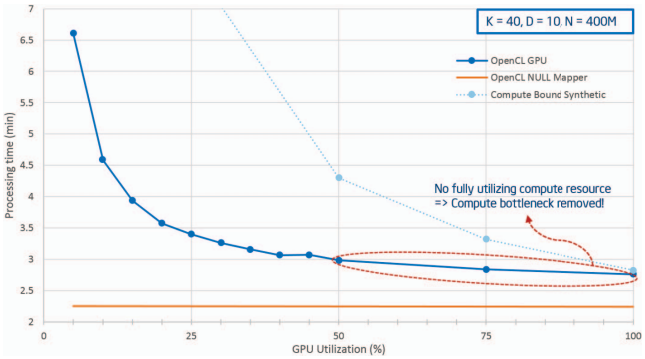
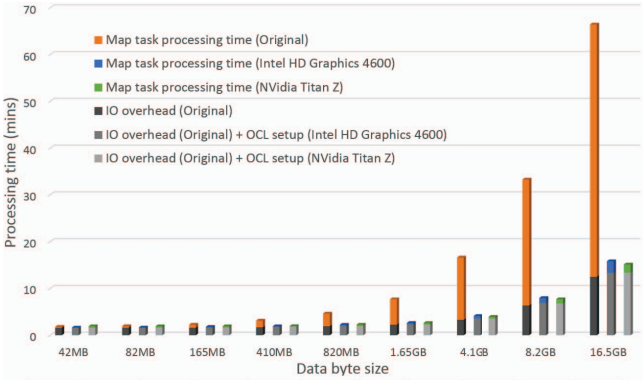


Fig. 5. Job processing time as a function of capping max allowed GPU utilization. Above 50% of GPU utilization, the performance has stabilized.



Performance gain		42MB	82MB	165MB	410MB	820MB	1.65GB	41GB	8.2GB	16.5GB
Map task-only	Integrated	4.62	6.28	3.82	10.00	8.89	17.00	24.30	25.11	21.82
	Discrete	0.63	1.07	2.31	16.06	8.56	15.81	36.02	29.02	30.75
Job	Integrated	1.09	1.17	1.26	1.62	2.06	2.92	4.02	4.18	4.20
	Discrete	0.93	1.01	1.17	1.60	2.06	2.94	4.22	4.32	4.39

Fig. 6. Comparison of total processing time for original CPU and OpenCL versions on integrated and discrete GPUs, split into IO overhead and Map task processing time. The table shows performance gains for Map task-only and Job processing time relative to original CPU version (1-node cluster).

lowest IPC value. The higher IPC value the workloads have, the more likely workloads are compute-intensive. Thus, previous studies justify that we use K-means clustering, a compute-intensive workload in HiBench, to embody our approach.

B. Experimental Design

Performance in Time: To evaluate performance gains in time, we measure total processing time of our cached CPU version and OpenCL GPU version, with different sizes of data. We then compare the processing time with that of the original CPU version. Further, to better understand the performance benefits of our OpenCL kernels for map tasks, we also observe a metric from the Hadoop Job Counters, *slots_millis_maps*, described as 'Total time spent by all maps in occupied slots'. To minimize runtime performance variation, we run each test multiple times and average the processing time if the samples are within 5% deviation.

IO Overhead Isolation: MapReduce is designed for processing extremely large datasets reliably. While increasing reliability, this paradigm also introduces significant IO overhead. To better understand the effect of IO on cluster performance, we conducted a few experiments to help us isolate the impact of the file system IO speed. To develop a baseline as to what our maximum possible performance increase over a standard CPU implementation would be, we performed what we called a NULL Mapper test. This establishes a lower bound on performance based upon how fast we can read data from HDFS. To perform the NULL Mapper tests we started by changing the Map function to be completely empty. We also turned off the Reduce step as we wanted to be able to time just how quickly we could send data to the Map step.

Power Consumption: We used the WattsUp [33] power meter device to measure the power consumption on our clusters. On a 4-node cluster, the meter is set up to capture the power consumption of all 4 nodes combined. The goal of this test was to verify that when using the integrated GPU for processing data, the energy consumption per unit time is not significantly higher than if it is running with CPU only. We also performed power consumption tests on our single node clusters, to capture the power consumption rates of CPU + integrated GPU and CPU + discrete GPU.

V. Results and Discussion

A. Job Processing Time

With K being the number of clusters in the K-means algorithm, and D the dimensionality of the input points, Fig. 2 shows the performance gains of our cached CPU version and OpenCL GPU version against the original K-means with two different data configurations, (K=20,D=20) and (K=40,D=10). In both configurations, our OpenCL version outperforms the original K-means for input data sizes greater than 165MB. Since the data generated with higher K introduces more computation into the OpenCL kernels, the performance gain increases with K growth (2.97x for K=20, 4.37x for K=40). Whereas the performance gain for the cached CPU version does not show a significant difference between these two configurations, indicating that the CPU version is compute bound. As such, we verify that the huge benefit comes from the increased compute offered by the GPU rather than as a by product of the caching mechanism.

B. Map Tasks Time

While the performance is evaluated in the previous section for the whole K-means run, we seek to understand the distribution of processing time for each component in more detail. We run NULL Mapper tests as described in Sec. IV-B for various datasets with (K=40, D=10). Fig. 3 shows the result of the processing time being split into IO overhead, and Map tasks-only processing time. It is observed that the IO overhead, which includes creation, distribution and tracking of tasks, and network/disk IO, remained the same for both the original CPU and OpenCL GPU versions. However, the Map task processing time was reduced drastically with our combined optimization of caching and GPU acceleration. It reduced from originally being ≈83% of the total processing time, to constitute only ≈20% after the acceleration. The overall performance gain was ≈4.37x for the largest input data size shown in Fig. 6.

If the analysis is focused on the performance of map processing time only, rather than the total processing time,

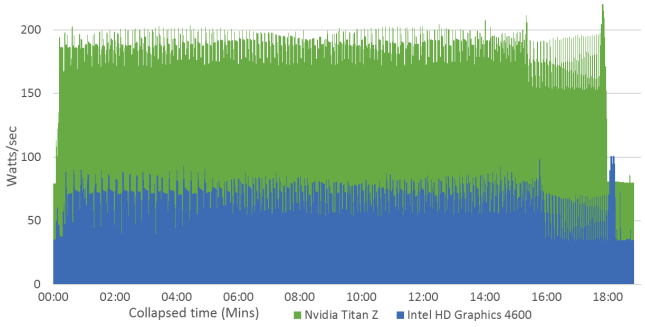


Fig. 7. Power consumption for our OpenCL version of K-means on discrete vs. integrated GPU. Experiment was performed on single-node cluster with data configuration of (K=40, D=10, Data size=16.5GB).

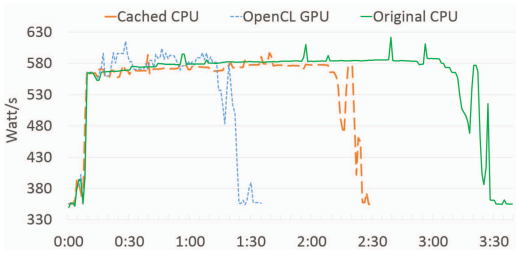


Fig. 8. Power consumption for K-means in original, cached CPU and OpenCL GPU versions (4-node cluster) with data configuration of (K=40, D=10, filesize=4GB).

TABLE II. DATA PROCESSING THROUGHPUT NORMALIZED TO POWER CONSUMPTION.

	KB/sec/Watt
OpenCL GPU version on Intel HD 4600	224
OpenCL GPU version on NVidia Titan Z	86
Original CPU version	60

the gain ratio is much higher for the OpenCL GPU version over the original CPU. Fig. 4 shows the ratio for map only processing time, for various K and D values on our 4-node cluster. For fairness of evaluation, the total file size of the input data is kept constant for all test cases, that is, for an increased D value, the number of input points is reduced proportionally to keep the input data size constant. While for all configurations we observe positive gains, the amount of improvement increases with increasing K, but with decreasing D. This can be explained as the following: 1) As K increases, so does the intensity of compute in the workload. Thus there is more benefit to offloading the compute to GPU. 2) As D decreases, the cache buffer contains more data elements, resulting in increased performance. Whereas, smaller D makes data more fragmented for original CPU version, which increases the amount of IO for each data element.

C. IO Bottleneck

With the NULL Mapper test, we also performed experiments to determine the effect of available compute on processing time as well as to understand the theoretical optimal bound. In Fig. 5 we show the results of this experiment in which we kept the workload constant but limited the maximum potential GPU utilization. The orange line on bottom is the NULL mapper test time and remains constant as expected. The solid blue line is our K-means workload and the dotted light blue line is a synthetic heavily compute bound kernel. Analyzing these results we see that the execution time of our OpenCL accelerated jobs stay mostly flat when the GPU utilization has reached 50% or beyond. In contrast the more compute intensive workload shows a much steeper drop off in performance as a function of available compute. From this we speculate that we have surpassed the point of being compute bound and are now strictly IO bound on our current test systems.

D. Integrated vs. Discrete GPU

If the hypothesis that we successfully transformed a compute-bound workload into an IO-bound workload is correct, then we would expect to observe very little increase in performance, even if more compute power is added to the test cases. As such, we performed experiments to support our hypothesis by adding a powerful discrete GPU (NVidia Titan Z 705MHz) to one of our single-node clusters. The performance comparison is shown in Fig. 6, in which we observe that Job processing times are similar with performance gain of 1.09-4.2x for the integrated GPU and 0.93-4.39x for the discrete GPU. This is because the more we improve the performance of Map tasks with OpenCL, the more IO overhead dominates the total Job processing time as per our hypothesis. For Map tasks-only, performance gains are 3.82-25.11x for the integrated GPU and 0.63-36.02x for the discrete GPU. As shown in Fig. 6, both Job and Map task-only processing times on the integrated GPU outperform the original CPU version for all test data, whereas on the discrete GPU they outperform from the second smallest data.

E. Power Consumption and Efficiency

Since discrete and integrated GPU present similar performance, their next differentiating factor is power consumption. Fig. 7 illustrates a power consumption measurement for both test configurations. For this test run the discrete GPU consumed nearly 3x more power than the integrated GPU. In addition, the idle state of the two systems also shows the discrete GPU system consuming significantly more power than the system utilizing the integrated GPU. We also compare power consumption of the original CPU, cached CPU, and OpenCL GPU running on the integrated GPU as shown in Fig. 8. The results show that with the utilization of the integrated GPU, there is no evident increase of power consumption per unit time. As the processing time has shortened by a factor of 2.9, the overall consumption has decreased as well. For this same experiment, we have converted the data into power efficiency values, which are shown in table Table II. Although the NVidia discrete card presented similar processing performance compared to Intel HD 4600, its overall processing efficiency per Watt consumption (KB/s/Watt) is much lower. In this particular case, the processing efficiency of Intel HD graphics presents a 2.6x gain over NVidia, and the OpenCL GPU accelerated method presents a 3.7x processing efficiency gain over the original CPU version.

VI. CONCLUSION AND FUTURE WORK

We presented a scalable method which modifies Map tasks to enable acceleration using the integrated GPU through

OpenCL. We demonstrated its applicability through an in-depth study in the K-means algorithm (Apache Mahout implementation), and performed benchmarking through HiBench suite. We evaluated this workload using various metrics and under multiple environments. The timing data that we collected shows a clear speed-up and significant power reduction when utilizing the integrated GPU via OpenCL.

Further in-depth analysis on the GPU utilization, which presented stabilized time performance for ≈50%-100% GPU utilization, triggered our speculation that the compute-bottleneck had been transformed into an IO-bottleneck. Further experiments involving adding a powerful discrete GPU into the test system, showed similar performance results, which confirmed our thoughts that for the K-means workload, additional compute power from the integrated GPU had shifted the compute bottleneck to an IO bottleneck. We further demonstrate that although discrete GPUs can be used to achieve similar performance in time, their power consumption per unit time is much higher than integrated GPU. In conclusion, integrated GPU seems a favorable choice in terms of cost-performance balance for Data Centers, for workloads similar to K-means. Moreover, the intensive IO from the MapReduce paradigm limits the potential for performance boosts from powerful GPUs. In this example, the additional compute performance available from the integrated GPU was already sufficient to transform a compute intensive K-means algorithm into an IO-bound problem.

We do believe that our approach is generalizable, even though it requires per-workload modification to Map or Reduce tasks. Finally future work is also needed to test the effects of other system components and configurations to understand their impact on processing time and power efficiency. In particular we would like to test on systems with faster IO technologies, such as SSDs and/or systems providing larger bandwidth of disk IO.

REFERENCES

[1] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf, accessed: Feb. 2, 2015.

[2] http://hadoop.apache.org, accessed: Nov. 21, 2014.

[3] M. Xin and H. Li, "An implementation of gpu accelerated mapreduce: Using hadoop with opencl for data- and compute-intensive jobs," in *International Joint Conference on Service Sciences*, May 2012, pp. 6–11. [Online]. Available: http://dx.doi.org/10.1109/IJCSS.2012.22

[4] C. Basaran and K.-D. Kang, "Grex: An efficient mapreduce framework for graphics processing units," *Journal of Parallel and Distributed Computing*, vol. 73, no. 4, pp. 522–533, April 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2013.01.004

[5] S. Kim, I. Roy, and V. Talwar, "Evaluating integrated graphics processors for data center workloads," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, ser. HotPower '13. New York, NY, USA: ACM, 2013, pp. 8:1–8:5. [Online]. Available: http://doi.acm.org/10.1145/2525526.2525847

[6] E. Ching, N. Egi, M. Mortazavi, V. Cheung, and G. Shi, "Unleashing the hidden power of integrated-gpus for database co-processing," in *Jahrestagung der Gesellschaft für Informatik, Informatik*, Sept 2014, pp. 1755–1766. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings232/article290.html

[7] https://mahout.apache.org/, accessed: Nov. 21, 2014.

[8] M. Hammoud and M. F. Sakr, "Locality-aware reduce task scheduling for mapreduce," in *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 570–576. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.87

[9] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," in *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 40–47. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.16

[10] L. Chen, X. Huo, and G. Agrawal, "Accelerating mapreduce on a coupled cpu-gpu architecture," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 25:1–25:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389030

[11] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, "Shadoop: Improving mapreduce performance by optimizing job execution mechanism in hadoop clusters," *Journal of Parallel and Distributed Computing*, vol. 74, no. 3, pp. 2166–2179, March 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2013.10.003

[12] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A mapreduce framework on graphics processors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 260–269. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454152

[13] M. Elteir, H. Lin, W. chun Feng, and T. Scogland, "Streammr: An optimized mapreduce framework for amd gpus," in *Proceedings of the IEEE 17th International Conference on Parallel and Distributed Systems*, ser. ICPADS '11. Washington, DC, USA: IEEE Computer Society, Dec 2011, pp. 364–371. [Online]. Available: http://dx.doi.org/10.1109/ICPADS.2011.131

[14] Y. Zhai, E. Mbarushimana, W. Li, J. Zhang, and Y. Guo, "Lit: A high performance massive data computing framework based on cpu/gpu cluster," in *IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2013, pp. 1–8. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CLUSTER.2013.6702614

[15] A. Papagiannis and D. Nikolopoulos, "Rearchitecting mapreduce for heterogeneous multicore processors with explicitly managed memories," in *Proceedings of the 39th International Conference on Parallel Processing*, ser. ICPP '10. Washington, DC, USA: IEEE Computer Society, Sept 2010, pp. 121–130. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2010.21

[16] F. Ji and X. Ma, "Using shared memory to accelerate mapreduce on graphics processing units," in *IEEE International Parallel Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, May 2011, pp. 805–816. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/IPDPS.2011.80

[17] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++: Modular mapreduce for shared-memory systems," in *Proceedings of the 2nd International Workshop on MapReduce and Its Applications*, ser. MapReduce '11. New York, NY, USA: ACM, 2011, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/1996092.1996095

[18] L. Chen and G. Agrawal, "Optimizing mapreduce for gpus with effective shared memory usage," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '12. New York, NY, USA: ACM, 2012, pp. 199–210. [Online]. Available: http://doi.acm.org/10.1145/2287076.2287109

[19] R. Zheng, K. Liu, H. Jin, Q. Zhang, and X. Feng, "Accelerate mapreduce on gpus with multi-level reduction," in *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, ser. Internetware '13. New York, NY, USA: ACM, 2013, pp. 10:1–10:8. [Online]. Available: http://doi.acm.org/10.1145/2532443.2532447

[20] C. Hong, D. Chen, W. Chen, W. Zheng, and H. Lin, "Mapcg: Writing parallel program portable between cpu and gpu," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 217–226. [Online]. Available: http://doi.acm.org/10.1145/1854273.1854303

[21] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis,

"Evaluating mapreduce for multi-core and multiprocessor systems," in *Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture*, ser. HPCA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 13–24. [Online]. Available: http://dx.doi.org/10.1109/HPCA.2007.346181

[22] http://developer.amd.com/tools/heterogeneous-computing/aparapi, Sept 2011, accessed: Jan. 27, 2015.

[23] A. Hayashi, M. Grossman, J. Zhao, J. Shirako, and V. Sarkar, "Accelerating habanero-java programs with opencl generation," in *Proceedings of the International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*, ser. PPPJ '13. New York, NY, USA: ACM, 2013, pp. 124–134. [Online]. Available: http://doi.acm.org/10.1145/2500828.2500840

[24] P. Calvert, "Parallelisation of java for graphics processors," Master's thesis, University of Cambridge, 2010.

[25] P. Pratt-Szeliga, J. Fawcett, and R. Welch, "Rootbeer: Seamlessly using gpus from java," in *High Performance Computing and Communication IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, June 2012, pp. 375–380. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/HPCC.2012.57

[26] M. Grossman, M. Breternitz, and V. Sarkar, "Hadoopcl: Mapreduce on distributed heterogeneous platforms through seamless integration of hadoop and opencl," in *Proceedings of the IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, ser. IPDPSW '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1918–1927. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2013.246

[27] http://www.jocl.org/, accessed: Nov. 21, 2014.

[28] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," pp. 41–51, March 2010. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICDEW.2010.5452747

[29] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 1197–1208. [Online]. Available: http://doi.acm.org/10.1145/2463676.2463712

[30] J. Dai, "Performance, utilization and power characterization of hadoop clusters using hibench," in *Hadoop in Chian 2010*, 2010.

[31] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," *CoRR*, vol. abs/1307.8013, 2013. [Online]. Available: http://arxiv.org/abs/1307.8013

[32] F. Pan, Y. Yue, J. Xiong, and D. Hao, "I/o characterization of big data workloads in data centers," *Big Data Benchmarks, Performance Optimization, and Emerging Hardware Lecture Notes in Computer Science*, pp. 85–97, 2014. [Online]. Available: http://rd.springer.com/10.1007/978-3-319-13021-7_7

[33] https://www.wattsupmeters.com/, accessed: Nov. 21, 2014.