PICO: A MapReduce Framework for Mobile GPUs

Ahmad Rahman Abdul Basit Muhammad Usman Irfan Jawwad Ahmed Shamsi
Systems Research Lab, Department of Computer Science
National University of Computer & Emerging Science
{k122026, k122290, k122023, jawwad.shamsi}@nu.edu.pk

Abstract— **This paper presents our stand-alone MapReduce framework, PICO, that harvests the power of GPUs for data driven computation. Our work is inspired by higher available computation power of GPUs and by the fact that data-driven applications are being increasingly used. Motivated by the above mentioned facts, we implement PICO on NVIDIA's first stand-alone mobile GPU Jetson TK1. PICO is designed in a manner to maximize the GPU utilization by using large amounts of map and reduce items into chunks and using amassing and fractional reduction. We use two standard MapReduce benchmarks and compare the performance achieved by PICO with – a popular and existing GPU-based MapReduce library, to show the flexibility and supremacy of PICO. Our results confirm the efficacy of PICO and demonstrate higher speedups than existing MapReduce libraries.**

Keywords-GPGPU; Data Driven; MapReduce; Parallel Computing

I. INTRODUCTION

Massive utilization of data has created profound opportunities and highlighted new challenges for the research community. As data is being increasingly utilized for continuous analysis and to unleash many unresolved answers [2], data-driven computing [24] has gained massive popularity. The term data-driven computing refers to the type of computing in which huge amount of data is used to solve complex computational problems. With the increased popularity of data-driven computing, new platforms are needed which are capable to effectively meet the growing needs and challenges.

The foremost requirement for data-driven applications is to have the capability of extensive computational power in order to solve such applications in a timely manner. Further, infrastructure requirements such as large amount of memory and storage also exist.

Technological advancements have led to increased utilization of Graphic Processor Units (GPU) for computationally extensive tasks, including data-driven computing. While GPUs have the potential to provide high computational power coupled with increased memory, there are certain important considerations which are needed to be met. Most importantly, there is a need for an efficient framework which can cater the challenges of data-driven computing on GPUs. The framework must have following important features:

- Provide higher abstraction to the developer in order to hide the complexities of data-driven computing and GPU programming.
- Must have low synchronization overhead among multiple threads of GPU. This would enable high scalability.
- Should have efficient load-balancing scheme in order efficiently utilize the massive parallelism of GPU.
- Should have efficient mechanisms for read, writes, and file and string processing in order to facilitate reduced latency.

In this paper, we are motivated by the above mentioned needs and requirements. To this end, we propose and develop PICO - an efficient, Map-Reduce based framework for embedded GPUs. Development of PICO on GPUs provide increased computational power. Further, unlike existing GPU-based frameworks such as Mars [9], PICO has been leveraged through performance enhancement techniques such as amassing and fractional reduction. In the former case, key-value pairs emitted from multiple mappers are cascaded with each other before being sent to the Reducer. Whereas, in the latter case, output from multiple mappers are combined before being sent to the reducer. Both these techniques reduce memory bottleneck. Moreover, PICO exploit GPU parallelism at its maximum. It also caters synchronization and communication overheads by chunking data items and inter-thread cooperation mechanism.

We implement PICO on Jetson TK1 [23] - NVIDIA's first mobile GPU processor. We also compared the performance of PICO with Mars. For this purpose, we ported Mars for the embedded processor and compared its performance with PICO. Using Word count and Matrix multiplication as benchmarks, we observed that PICO provides a speedup for greater than 2. Following are the major contributions of this paper:

- Developed a MapReduce framework on the Jetson TK1, to increase the performance of the system to large scales on a power budget. The Jetson TK1 is a GPU based device regarded as the world's first embedded supercomputer by NVIDIA.
- Incorporated enhancements for the high parallelism and the memory optimizations for a high performance.

- Ported MARS on Jetson TK 1 and successfully achieved double speedup for popular benchmarks.

Our framework, PICO, also leverages the “CUDA Data-Parallel Primitives” library (CUDPP) [5], specifically its scan [6] and sort [7] primitives. Using CUDPP, we were able to design PICO more easily and focus on the API since CUDPP handles many of the difficult aspects

In this paper, we describe the design and implementation of PICO and share results of performance comparison with Mars. A novel contribution of this paper is to demonstrate efficacy of embedded GPUs for data-driven computing. We intend to provide PICO as an open source library to the community, which will enable new horizons of embedded GPU computing. We anticipate that this will be highly beneficial for the community.

The remainder of this paper is organized as follows. Section II explains the background and related work in the field. Section III explains the design and implementation of our framework. Section IV elucidates the experimental setup used for evaluations. Section V explains our findings and experimental results. Finally, we conclude our paper and propose our future work.

II. Background and Related Work

This section describes background about the MapReduce framework and Tegra Jetson TK1- NVDIA’s latest mobile supercomputer. The section also elaborates significant research by other researchers related to data-driven computing and utilization of GPUs for data-driven tasks.

A. *Background*

Now a days GPUs are commonly used for data driven approaches like the various sorting algorithms, usually data driven algorithms are fastest, which implies that the step the algorithm takes depends on the value of the key currently under consideration, for example the well-known Quick sort algorithm. General Purpose Computing on graphic processing units (GPGPU) is currently very common and recently used for various applications [22] such as matrix operations [18], FFT computation [17], embedded system design [10], bioinformatics [19] and database applications [13] [14]. Most of the existing work adapts or redesigns a specific algorithm on the GPU.

Consequently, to process such gigantic data collections for these applications high performance is essential [2]. Today, MapReduce is the most efficient framework to process big data [2] [14]. Moreover, due to the moderately small overhead and the considerable amount of parallelism, exploiting GPUs computation is becoming an important aspect for better performance and efficiency per unit price and energy.

Initially Google developed MapReduce for large-scale internal data processing which was very successful in terms of performance, time and cost, which popularized MapReduce architecture around the globe, and many MapReduce libraries and packages were developed.

MapReduce is a software architecture that amalgamates two functions namely, Map and Reduce, in programing language. The input to map function is raw, un-processed data, on which certain processing is applied and independent key value pairs are produced as output, the similar keyed values are grouped together and handed to a reduce function, where after certain computations result is produced. The Map and Reduce functions are written by application developers themselves.

Stimulated by the recent GPU computation trends and inherent parallelism, NVDIA introduced a 192-core Tegra embedded processor to harvest GPU computation at its maximum.

Tegra K1 is NVIDIA's first mobile processor to have the same advanced features & architecture as a modern desktop GPU while still using the low power draw of a mobile chip. Therefore, Tegra K1 allows embedded devices to use the exact same CUDA code that would also run on a desktop GPU (used by over 100,000 developers), with similar levels of GPU-accelerated performance as a desktop. Jetson TK1 is NVIDIA's embedded Linux development platform featuring a Tegra K1 SOC (CPU+GPU+ISP in a single chip), selling for $192 USD. Jetson TK1 comes pre-installed with Linux4Tegra OS (basically Ubuntu 14.04 with pre-configured drivers).

To best of our knowledge till present day there is no such MapReduce library that works on Jetson TK1, it has been used in various projects like emulators and robotics.

B. *Related Work*

We now briefly survey the techniques of developing GPGPU primitives, which are building blocks for other applications. Govindaraju et al. [13] presented novel GPU-based algorithms for the bitonic sort. Sengupta et al. [24] proposed the segmented scan primitive, which is applied to the quick sort, the sparse matrix vector multiplication. He et al. [15] proposed a multi-pass scheme to improve the scatter and the gather operations, and used the optimized primitives to implement the radix sort, hash searches.

Likewise, He et al. [16] further developed a small set of primitives such as map and split for relational databases. These GPU-based primitives improve the programmability of the GPU and reduce the complexity of the GPU programming. However, even with the primitives, developers need to be familiar with the GPU architecture to write the code that is not covered by primitives, or to tune the primitives. For example, tuning the multi-pass scatter and gather requires the knowledge of the memory bandwidth of the GPU [15]. In addition to single-GPU techniques, GPUs are getting more involved in distributed computing projects such as Folding@home [11] and Seti@home [25]. These projects develop their data analysis tasks on large volume of scientific data such as protein using the computation power of multiple CPUs and GPUs. On the other hand, Fan et al. [9] developed a parallel flow simulation on a GPU cluster. Davis

Traits	PICO	I-MR	CellMR	MARS	MapCG	MapD
Parallel Algorithm	✓	✗	✓	✓	✓	✓
Single Node Compatible	✓	✓	✓	✓	✓	✓
Multiple Node Compatible (Scalability)	✓	✓	✗	✗	✗	✓
User Control over scheduling threads, blocks etc.	✓	✗	✗	✗	✗	✗
Optimizations for Communication Overhead	✓	✗	✗	✗	✗	✗

TABLE I. Summary of Related Work in The Field

et al. [7] accelerated the force modeling and simulation using the GPU cluster. Goddeke et al. [12] explores the system integration and power consumption of a GPU cluster of over 160 nodes. As the tasks become more complicated, there lacks high-level programming abstractions to facilitate developing these data analysis applications on the GPU.

I-MapReduce [10] (formerly CGL MapReduce) is another MapReduce library like MARS, and theoretically the first to ensure the use of data streams instead of hard disk access. Intermediate key value pairs and reduce function outcomes are buffered directly to new mapper and reducer nodes for further computation. By which an efficient, iterative MapReduce algorithm with multiple consecutive MapReduce processes was developed.

Recently, paradigm has shifted towards MapReduce to be executed on parallel processors resembling GPUs and IBM's Cell. Catanzaro et al. developed a single-node library for GPUs [11] but the emphasis was on numerous insignificant jobs. Mars [9] was the first significant GPU structure, though its scalability is restricted; for execution Mars uses only one GPU and in-GPU-core tasks. Additional deficiency is that the package, schedules threads and blocks, user has no control over it, making it hard to entirely exploit certain GPU proficiencies (e.g. inter-block communication). CellMR [13] is a single-node MapReduce library on the Cell Engine that improves the in-core problem of Mars by issuing map function's data in minor portions. In CellMR the MapReduce architecture is distributed into three main portions: Map, Partial Reduction and Global Reduction. Moving on MapCG [12] is another GPU-based MapReduce library. The key objective was to allow for portable multicore MapReduce code that could execute on the GPU. Like Mars, MapCG also has restricted scalability because it uses only one GPU. MapD, exploits the immense computational power available in off-the-shelf graphics cards that can be found in any laptop or PC. But MapD is strictly suitable for data analysis, machine learning, real-time querying, and data visualization, and can cater specific application use cases. Table I further summarizes the comparison between our framework and related libraries.

III. DESIGN AND IMPLEMENTATION

This section highlights the major issues and challenges specific to GPU implementation of MapReduce. It also elaborates our design choices in dealing with these issues.

A. *Design Considerations and Features*

The single in-core CPU MapReduce implementation is easy. A chunk of indivisible tasks is mapped on that single CPU, which generates key-value pairs. Than the key-value pairs are sorted and passed to a reducer which provides the resultant desired output. Transforming this implementation to GPU (Jetson TK-1) - embedded processor with limited memory, is inspired by the following design considerations and features:

1) *Shared Memory between CPU and GPU.*

In contrast to current desktop / server GPUs, the memory on Tegra is physically unified. This implies that CPU and GPU share memory with different logical address space. Shared memory helps in reducing memory latency by decreasing communication overhead. As same chunk of memory can be used by both CPU and GPU.

2) *Partioning of output.*

As Jetson TK1 is an embedded system, it has limited memory. This implies that efficient utilization of memory is needed in order to get good efficiency.

To cater this limitation, we partition the output of the Map phase. This partitioning implementation improves the efficiency such that data items do not exceed the memory size limit.

3) *Processing data items in small groups*

PICO partitions the input data into small groups called chunks. This approach is beneficial for multiple reasons. First, every GPU thread knows about every other GPU thread's working data items. Additionally, processing a group of task concurrently makes our architecture more flexible, fast and efficient. It also provides high level of abstraction from the original data. Furthermore, it allows developers to exploit GPUs at full capacity (for example, manually schedule threads per block etc.). But all this comes with a tradeoff between abstraction and performance. It also allows us to Map or Reduce concurrently another chunk while copying another chunk to or from GPU.

4) *Maximum utilization of inherent paralellism on GPU*

As Tegra K1 has 192-cores, we would like to exploit maximum parallelism in solving MapReduce tasks. In order to achieve this, Map and Reduce are distributed across cores.

5) *Inter-thread cooperation*

We want to allow programmers to optimize particular workloads and exploit complete GPU strength with less PCI-e overhead. We have exposed partitioning and sorting/grouping to the programmer. Moreover, we have relaxed the mapping of one item to one thread. Due to this many-to-many and one-to-many mapping of data items to kernel thread, inter-thread cooperation is achievable.

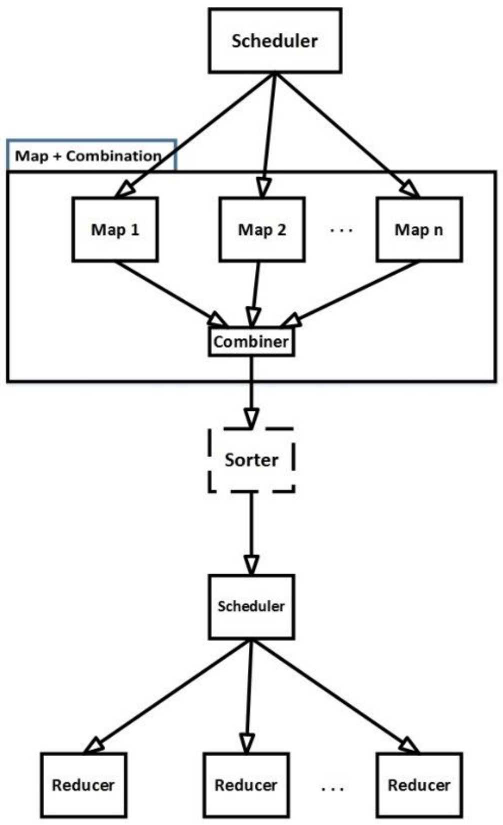

Figure 1. Architecture of MapReduce Combine Phase. This optimization is a variant of an existing MapReduce task, Combine.

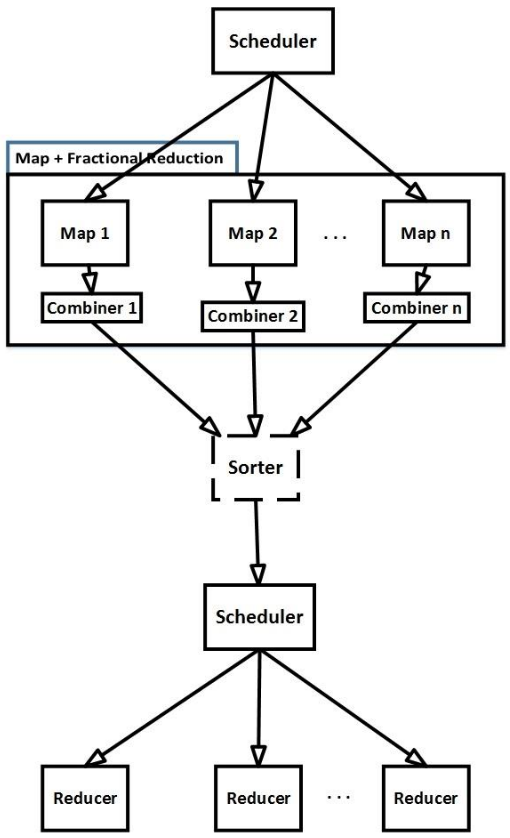

Figure 2. Architecture of Fractional Reduction Optimization

This also helps in efficient higher-level operations and provides flexibility to the user.

6) Overlapping communication overhead

The key to achieve parallel efficiency is to reduce communication overheads and overlap them with computation and useful processing. We know that communication is always a bottleneck and GPU processing is comparatively easy and less over-headed, so we introduced some optimizations to achieve computation over communication listed below.

- Combine
- Amassing
- Fractional Reduction

a) Combine Phase

After Map stage when key-value pairs are produced we group same key value pairs before passing them to Reducer. The Grouping doesn't generate new keys but instead combine like keys to assign them a single grouped value, in-turn reducing multiple combinations. To reduce network communication, we typically only group values from the same Map instance.

b) Fractionl Reduction

The main goal of Fractional Reduction is to eliminate communication overheads, overlapping it with maximum GPU computation. To mitigate certain in-memory communication cost, the user can process Fractional Reductions on GPU-resident key-value pairs to group like-keyed pairs, resulting in less pairs and reduced communication overhead.

c) Amassing

Amassing is nearly similar to Fractional Reduction; its main goal is to decrease communication overhead and to reduce the key-value pairs. Amassing provides knowledge to each mapper about the key-value pairs, when intermediate key-value pairs are generated that remain on GPU and each subsequent Map kernel combine its resultant to those intermediate key-value pairs by adding new pairs, grouping like pairs or reducing number of pairs.

On a broad-spectrum, Fractional Reduction is more beneficial when the estimated final key-value set is enormous, and Amassing should be used when the estimated final key-value set is trivial.

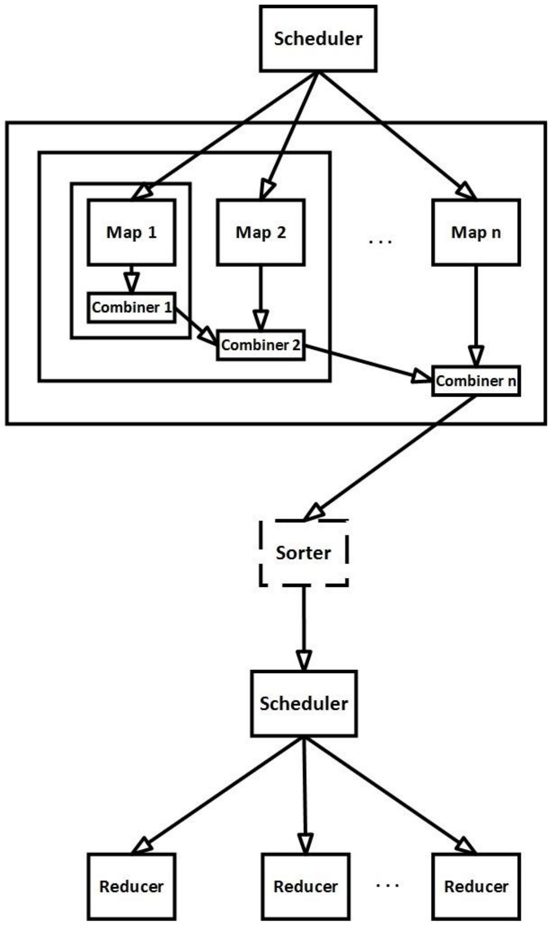

Figure 3. Architecture of Amassing Optimization

B. *Implementaation Details*

We developed PICO, in C++ and CUDA with an aim of ease-of-use and extendible, while still permitting exploitation of GPU. As illustrated above, we have provided default implementations of sorter. However, to provide flexibility and ease of use to the user every stage of our frameworks' pipeline is programmable by the user.

There are three separate stages of our MapReduce framework namely, Map, Sort, and Reduce. Unlike Map, Sort and Reduce are indivisible whereas, Map is further divided into numerous distinct sub-stages. Every process executes a MapReduce pipeline and each GPU is controlled by a separate process. These three stages are further described below.

1) *Map Stage*

The Map stage takes an input chunk and after processing outputs a set of key-value pairs. The complete chunk is cached into the GPU by a programmer specified method atomically. It is then processed by one or more programmer specified kernels. As described earlier the Map stage is divided into several sub-stages and can take many type of forms upon the user's choice of optimizations (Map itself, Amassing and Fractional Reduction). Map executes single chunk at a time as our framework, PICO, assumes that whole GPU memory will be occupied by that chunk's processing and output. This architecture permits the raw usage of the GPU while still upholding the MapReduce architecture of data-independent elements.

2) *Sort Stage*

The Sort stage is reasonably simple and indivisible. Radix sort is provided as the default sorter for integer based key-value pairs, for other type of keys the user has to define his own sorting implementation. PICO discards similar keys and every key's values are stored contiguously as a result of sorting. The sorting stage is designed to be as flexible as possible with maximum functional exploitation of GPU. Thus, user only requires the number of values and the index of the first value define each arrangement.

3) *Reduce Stage*

The Reduce stage is also reasonably simple and indivisible. In a similar fashion like Map, key-value pairs are divided into chunks in a bespoke manner, the chunks are formed according to the size of GPU memory. Our framework, PICO, executes some function calls in which it gathers the information that how many pairs should be copied to the GPU for next reduction step. These calls are issued until PICO processes the final key-value pair.

4) *Mapping Applications to PICO*

Most commonly, PICO can be used for same purposes as that of a CPU-based MapReduce framework. PICO is designed to be as flexible as possible. The developer just has to provide a Mapper and optionally a Reducer, and input data.

As described before PICO is completely customizable by the programmer. User can provide their own implementation of Sorter according to their applications. This provides an advantage of performance and scalability. However, PICO does provide a default implementation of integer-based sorter.

To increase performance and scalability, there are some obligations upon user. For example, they should use additional Map sub-stages and optimization according to their applications need. They should configure their flow of application to be as GPU bound as possible. Also, they should ensure computation over communication.

IV. Experimental Setup

We now describe the experimental configurations and the benchmark used for testing our framework.

A. *System Configuration*

To test our framework, PICO, we used Jetson TK1, at the National University of Computer and Emerging Sciences, FAST. The architecture comprises of Tegra K1 SOC with NVIDIA Kepler GPU with 192 CUDA Cores, NVIDIA, 4-Plus-1™ Quad-Core ARM® Cortex™-A15 CPU with 2 GB x16 Memory, 64-bit Width and 16 GB 4.51 eMMC Memory. Jetson TK1 comes pre-installed with Linux4Tegra OS (basically Ubuntu 14.04 with pre-configured drivers) and has CUDA 6.0.

We have evaluated our framework with the Word Count (WC, counts the number of times each word occurs in a text corpus) and Matrix Multiplication (MM) benchmark. These are considered as classic benchmarks for testing new MapReduce frameworks.

B. *Benchmark Guidelines*

Scalability is a factor that is most important for any MapReduce Application, and MapReduce libraries are reasonably good in parallelizing and scaling algorithms that are already scalable. Our framework is capable of that too. However, the most important question is to evaluate the performance of any framework on algorithms that are not perfectly scalable.

Therefore, we have preferred a benchmark that fail to scale for a diverse set of reasons and analyze its performance. Moreover, we further analyze other bottlenecks and potential performance issues in our framework, to measure the impact of these issues we needed to evaluate our framework in diversified aspects. Below we list some aspects and notify why they are important for consideration and how our benchmarks (Word Count, Matrix Multiplication) stressed that aspects.

- Expandability, if task is computation extensive — A computation extensive task can affect the scalability, if the internal configurations of framework does not handle it accordingly or it incurs any overheads. MM evaluates this characteristic as it is highly computation bound task.
- Varying number of emits for each thread — Possibility of emitting different number of key-value pairs by different maps of same chunk. WC evaluates this trait.
- Several emits for each thread — There is a situation when application emits more than one key for each map instance. It is easy for a CPU to cater, but takes additional preparation with a GPU. WC evaluates this characteristic.

C. *Benchmark Implementations*

We have implemented two benchmarks, Word Count and Matrix Multiplication to evaluate our framework. These are described below.

1) *Word Count (WC)*

Word Count. WC is a simple application that counts all incidences of a distinctive item. the items can be words, not integers or phrases. Input to the application is a file/corpus with numerous amount of words separated by spaces. Moreover, the result set for WC is much reduced, leading to an altered configuration of the pipeline and considerably diverse scaling. For our evaluations we have used a simplified dataset of Reuters, each chunk contains thousands of millions of bytes.

Distributing one line of text to GPU mappers is acceptable but there is a potential problem in mapping of task to GPU with this application, i.e. that we can’t use strings as keys, because length of string varies so they cannot be read in a single instruction, using variable-sized keys increases overheads of time and space and implementing fixed size string will surely yield poor storage performance in certain scenarios.

Instead, we used a minimal perfect hash [21] to assign each key a distinctive, four-byte integer value. As a result, the GPU Map kernel gives each thread one line of text and scans the text for words, then finishes by hashing W and emitting hash(W).

As our corpus is very simplified and small, so we have opted to use Amassing. By using Amassing, we diminish PCI-e transfer overhead and nearly remove all communication bottlenecks. Initially all keys are initialized to value 0, but every time we emit a key-value pair, we increment the value of each key according to its occurrence. There is no need to use Fractional Reduction because for now we are sending all key-value pairs to a single node, which is very fast on a small number of GPUs as a single Reduce kernel will handle the task. For Sorting, we have used the default sorter provided in our framework.

2) *Matrix Multiplication (MM)*

One of the most computation extensive and highly expandable on GPU benchmark is Matrix Multiplication (MM). The standard CPU based MapReduce MM algorithm, multiply matrices using N^2 vector-vector multiplications which have some short comings when used with a GPU. Like, it only works good on a GPU if we are reading row vectors of a matrix, GPU can’t hold row and column vector in its memory if matrices are too large.

Instead we use a structured approach from algorithms that are unaware of cache [20]. In our algorithm we split each matrix into minor portions up until every block in the GPU is according to the size of shared memory. Then every block performs a matrix multiplication where part of result is computed by each thread as an inner product. The Sort and Reduce stages are omitted in MM and we accumulate individual results in an another Map. This approach allows high scalability without imposing any particular thread-task mapping.

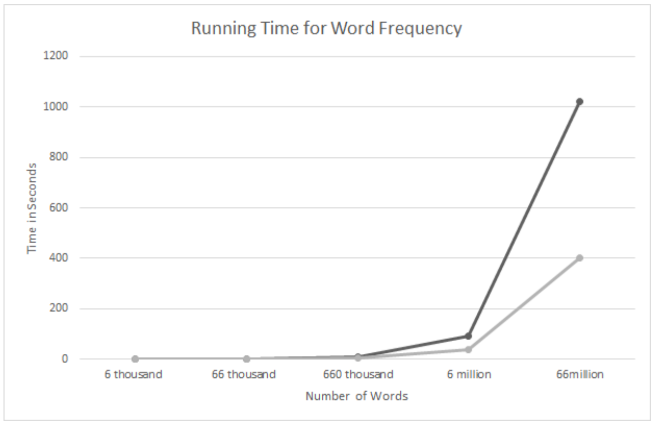

Figure 4. Word frequency running times for three subsets of dataset with both PICO and Mars on single node Jetson Tk1

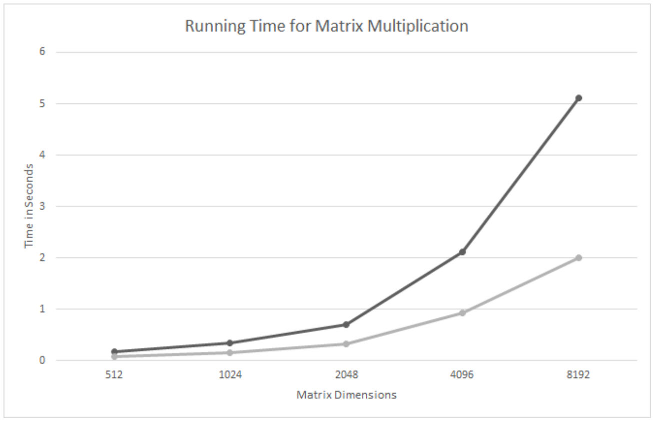

Figure 5. Matrix Multiplication running times for three subsets of dataset with both PICO and Mars on single node Jetson Tk1

V. Results

Using PICO, we have written two benchmark applications, word count or word frequency and matrix multiplication. We have used different subsets of overly simplified data set from Reuter's for word count. For comparison with PICO, we use the best known GPU-MapReduce framework Mars [9].

Fig. 4 shows the runtime breakdowns for our Word Count benchmark. The figure shows that our framework performs better than MARS in terms of execution time.

For matrix multiplication, we used three different subsets changing the dimensions of the matrices. We have used 1024^2, 2048^2 and 4096^2 as shown in figure 5.
The results show that PICO has a speedup of up to 2.47 over Mars with same datasets for the benchmark, word count and a speedup of 2.12 for the benchmark, matrix multiplication as shown in figure 5.

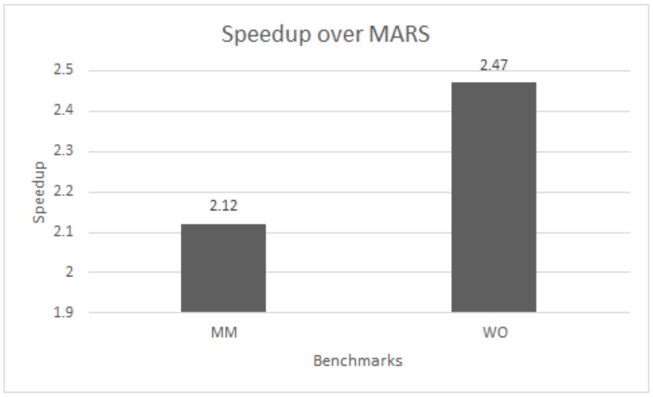

Figure 6. PICO Speedup over MARS

VI. Conclusion And Future Direction

In the recent era GPUs have become proficient accelerators for high performance computing. This paper proposes a framework, PICO, which exploits the computation power of mobile GPUs to speed up the MapReduce framework and at the same time, PICO provides enormous amount of flexibility to the developers specially compared to MARS, providing complete customizability of the MapReduce stages, while still delivering the default implementations. PICO delivers numerous other advantages to MapReduce programmers from which the most significant ones are the new optimizations provided by PICO to overcome communication overheads, scalability, flexibility and ease-of-use to the programmer. The insignificant overhead and communication cost of PICO makes it well qualified, compared to other MapReduce frameworks.

The key advantage of exploiting GPU over the CPU is mainly computation extensive tasks, MapReduce jobs require substantial computation to acquire the advantages through PICO. Our results demonstrate that like other MapReduce frameworks, PICO is unable to scale communication-bound tasks, that have extensive overheads of network, disk and PCI-e communication. Moreover, results conclude that the users of PICO should dedicate some of their time to decide what stages and optimizations (Amassing, Fractional Reduction) are appropriate for their MapReduce tasks, because these traits are dependent solely on the job at hand, together with its data sizes/input data size, computational complexity of the Map and Reduce.

For future, we will be developing a cluster of Jetson Tk1, using CUDA-aware MPI. We plan to evaluate the scalability and speedup of our framework and compare it with other MapReduce libraries.

Acknowledgment

This research has been supported by NVIDIA Teaching and Research Center Grants.

REFERENCES

[1] M. Ekman, F. Warg, and J. Nilsson, "An in-depth look at computer performance growth," ACM SIGARCH Computer Architecture News, vol. 33, no. 1, pp. 144– 147, Mar. 2005.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," ACM Queue, pp. 40–53, Mar./Apr. 2008.

[4] D. B. Kirk and W. W. Hwu, Programming Massively Parallel Processors. Morgan Kaufmann, 2010.

[5] M. Harris, J. D. Owens, S. Sengupta, Y. Zhang, and A. Davidson, "CUDPP: CUDA data parallel primitives library," 2009, http://gpgpu.org/developer/cudpp/.

[6] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," in GPU Gems 3, H. Nguyen, Ed. Addison Wesley, Aug. 2007, ch. 39, pp. 851–876.

[7] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore GPUs," in Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, May 2009.

[8] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multicoreandmultiprocessorsystems,"inInternationalSymposium on High-Performance Computer Architecture. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 13–24.

[9] 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, Oct. 2008, pp. 260–269.

[10] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," in Proceedings of the First International Conference on Cloud Computing, Oct. 2009.

[11] B. Catanzaro, N. Sundaram, and K. Keutzer, "A Map Reduce framework for programming graphics processors," in Third Workshop on Software Tools for MultiCore Systems, Apr. 2008.

[12] 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, Sep. 2010, pp. 217–226.

[13] M. M. Rafique, B. Rose, A. R. Butt, and D. S. Nikolopoulos, "CellMR: A framework for supporting MapReduce on asymmetric Cell-based clusters," in Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, May 2009.

[14] H. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-Reduce-Merge: Simplified relational data processing on large clusters," in SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, Jun. 2007, pp. 1029– 1040.

[15] C. Jin and R. Buyya, "MapReduce programming model for .NET-based cloud computing," in Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing. Berlin: Springer-Verlag, Aug. 2009, pp. 417–428.

[16] S. Chen and S. W. Schlosser, "Map-Reduce meets wider varieties of applications," Intel Research Pittsburgh, Tech. Rep. IRP-TR-08-05, May 2008. [Online]. Available: http://www.pittsburgh. intel-research.net/~chensm/papers/IRP-TR-08-05.pdf

[17] J. H. C. Yeung, C. C. Tsang, K. H. Tsoi, B. S. H. Kwan, C. C. C. Cheung, A. P. C. Chan, and P. H. W. Leong, "Map-Reduce as a programming model for custom computing machines," in FCCM '08: Proceedings of the 2008 16th International Symposium on Field-Programmable Custom Computing Machines, Apr. 2008, pp. 149–159.

[18] R. Lämmel, "Google's MapReduce programming model—revisited," Science of Computer Programming, vol. 68, no. 3, pp. 208–237, Oct. 2007.

[19] Y. Gu and R. L. Grossman, "Sector and sphere: the design and implementation of a high-performance data cloud." Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, vol. 367, no. 1897, pp. 2429–2445, Jun. 2009. [Online]. Available: 10.1098/rsta.2009.0053

[20] H. Prokop, "Cache-oblivious algorithms," Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Jun. 1999.

[21] R. J. Cichelli, "Minimal perfect hash functions made simple," Communications of the ACM, vol. 23, pp. 17– 19, January 1980.

[22] T. Aila and S. Laine, "Understanding the efficiency of ray traversal on GPUs," in Proceedings of High Performance Graphics 2009, Aug. 2009, pp. 145–149.

[23] Nvidia.com, "Realize Your Vision: World's First Embedded Supercomputer", 2014. [Online]. Available: https://www.citethisforme.com/guides/ieee/how-to-cite-a-website. [Accessed: 23- 5- 2015].

[24] Stutz, Michael (September 19, 2006). "Get started with GAWK: AWK language fundamentals". developerWorks. IBM. [Accessed: 2- 6-2015].