Tiny GPU Cluster for Big Spatial Data: A Preliminary Performance Evaluation

Jianting Zhang
Dept. of Computer Science
The City College of New York
New York, NY, USA
jzhang@cs.ccny.cuny.edu

Simin You
Dept. of Computer Science
CUNY Graduate Center
New York, NY, USA
syou@gc.cuny.edu

Le Gruenwald
Dept. of Computer Science
The University of Oklahoma
Norman, OK, USA
ggruenwald@ou.edu

***Abstract*—GPU-equipped computing nodes have much higher ratios between floating point computing power (in the order of TFlops and is fast growing) and network bandwidth (in the order of Gbps and remains stable) than regular computing nodes at which Hadoop-based systems are targeting. The gap makes efficient and scalable processing of large-scale data challenging, especially for geo-referenced spatial (or geospatial) data, whose processing is both data intensive and computing intensive. We aim at developing a tiny GPU cluster using Nvidia Tegra K1 (TK1) System on Chip (SoC) boards as a downscaled, low-cost GPU cluster for Big (Spatial) Data research. The tiny GPU cluster is equipped with standard gigabyte Ethernet network while has much less computing power and energy footprint when compared with a regular GPU cluster and represents a new platform with more balanced compute to communication ratio. We have ported our implementations of both single-node technologies for point-in-polygon test based spatial joins and the lightweight distributed execution engine originally developed for regular clusters to the tiny GPU cluster. We evaluate its performance on two real world geospatial applications with various settings and experiment results have demonstrated good scalability. Preliminary analysis on the scaling effect between the tiny cluster and a regular Amazon EC2 cluster using a simplified model suggest that the ARM-based CPU of the TK1 board is likely to achieve better energy efficiency while the Nvidia GPU of the TK1 board might be less efficient when compared with desktop/server grade GPUs, in both standalone and 4-node cluster settings.**

Keywords—Lightweight, Distributed Computing, GPU

I. INTRODUCTION

Hardware architectures and platforms have been evolving fast in the past few years, which have significant impacts on processing large-scale data. While Big Data software packages, such as Hadoop, were initially developed for inexpensive commodity workstations, as multi-core machines equipped with large memory capacities and hardware accelerators are becoming increasingly affordable, new Big Data systems that can take advantages of new hardware features and deliver high performance, such as Apache Spark[1] and Cloudera Impala [2] for in-memory and in-network processing, are becoming more preferable. As a result, there are growing interests on using High Performance Computing (HPC) facilities that are typically equipped with powerful processors (including accelerators) and high speed networks for Big Data applications[1][2][3][4]. Unfortunately, accesses to HPC facilities are very often restrictive and it is very difficult (if not impossible) to reconfigure HPC platforms for research purposes. On the other hand, while Cloud vendors typically allow users to choose among a few predefined Cloud resource configurations and install additional software stacks on top of either bare metal or virtual machines, the allocated computing instances do not allow hardware reconfigurations for experiments either. For example, only very few Amazon EC2 instances are equipped with Graphics Processing Units (GPUs) and the only type of GPU instances provided by Amazon EC2 is g2.2xlarge[3]. Furthermore, neither the details of the underlying physical GPU specification nor the virtualization overheads are specified. While this may be acceptable for production use, is inappropriate for research purposes. It is thus desirable to have a dedicated, fully configurable and high-performance cluster for Big Data research. Unfortunately, the Total Cost of Ownership (CTO) of such a cluster is typically high and the approach may not be always feasible.

GPS devices and smartphones have generated huge amounts of location data. Very often point location data need to be joined with urban infrastructure data to understand the location data and make better decisions. While geospatial data management techniques have been provided by both Spatial Databases[4] and Geographical Information Systems (GIS[5]), existing software is incapable of processing large-scale geospatial data for practical applications [5]. Quickly evolving processor, storage and networking technologies require new Big Data research to understand how new hardware impacts the performance of large-scale data processing. We have been developing techniques to process large-scale geo-referenced spatial (or geospatial) data on both single computing nodes and clusters equipped with GPUs and we refer to [5][6][7][8] for details. While a small GPU cluster with ~10 nodes has been built for internal use, the heterogeneity of the cluster with different generations of CPUs and GPUs, different memory capacities and different combinations of HDD/SSD drives has made it difficult to use. More importantly, GPU-equipped computing nodes have much higher ratios between floating point computing power (in the order of TFlops and fast growing) and network bandwidth (in the order of Gbps and remains stable in the past decade) than regular computing nodes at which Hadoop-based systems are targeting. While many research works have exploited high-performance

networks (e.g., Infiniband[6]) to narrow the gap and achieve better performance for Big Data systems, in this study, we propose to develop a tiny GPU cluster with much less computing power while being equipped with standard gigabyte Ethernet network, to investigate several Big Data research issues from a domain-specific application perspective. In particular, we have ported our implementations of both single node spatial data processing techniques and a lightweight distributed execution engine originally designed for regular clusters (referred to as LDE hereafter [8]) to the tiny GPU cluster and evaluate its performance on real world geospatial datasets. The performance is further compared with SpatialSpark [6], another distributed geospatial data processing system we have developed on top of Apache Spark, on the same tiny GPU cluster whereas appropriate.

Our technical contributions in this study can be summarized as follows. First, we report our effort in building a tiny GPU cluster consists of multiple Nivida Tegra K1 (TK1) System on Chips (SoC) boards[7], standard networking devices and Solid State Drives (SSDs) for Big (Spatial) Data research. The scaled-down GPU cluster has desired features for both data intensive and computing intensive applications that may not available in regular clusters, including high network bandwidth to compute ratio, GPU acceleration using the standard Nvidia CUDA[8] technology and shared memory between CPU and GPU. Second, we have ported our in-house developed LDE engines (including LDE-MC for multi-core CPUs and LDE-GPU for GPUs) and SpatialSpark (for multi-core CPUs) to the tiny GPU cluster. To the best of our knowledge, we are not aware of previous works on processing large-scale geospatial data on embedded systems (such as Tk1) with GPU accelerations in a cluster computing setting. Third, we report our experiment results on two real world geospatial applications and the results have demonstrated good scalability on the tiny GPU cluster.

II. Background and Motivation

The success of Hadoop-based systems has attracted quite some interests to improve Hadoop performance from many aspects, including porting it to HPC facilities to utilize their high-end computing processors, large memory capacities and high-speed networks. A comparison of architecture and abstractions between HPC and Apache Big Data Stacks (ABDS) is presented in [1] and the authors argued that a convergence between the two at many levels can be observed. While regular Hadoop uses the Java-based Netty[9] package for distributed communication, several works have proposed to use Message Passing Interface (MPI[10]) libraries, which are typically C/C++ based, to achieve better performance, especially on HPC clusters with high-speed networks [9][2]. A comprehensive assessment on the performance impact of high-speed interconnects (including 10Gbps Ethernet and Infiniband) on MapReduce is presented in [3]. A design of Hadoop Distributed File System (HDFS[11]) using Remote Direct Memory Access (RDMA[12]) over InfiniBand via Java Native Invocation (JNI[13]) is presented in [4] and a similar idea is also presented in [10]. Recently, [11] discussed how traditional HPC facilities can be optimized to accommodate both traditional HPC applications (computing intensive) and new data analytics applications (data intensive). While high-speed networks in HPC facilities have been demonstrated to achieve different levels of speedups over commodity Cloud resources, local storage is also crucial to the performance of Hadoop-based systems on HPC facilities. Unfortunately, traditional HPC facilities generally rely on dedicated storage nodes running parallel/distributed file systems (e.g., Luster-based[14]) which can be a bottleneck for data-intensive applications. Using HPC clusters for Big Data applications may require significant architectural redesigns to maximize efficiency, which further mandates novel ideas on setting up realistic yet low-cost and easy-to-access experiment environments. We believe our idea on using SoC clusters as scaled-down prototype systems could be an interesting idea.

Balancing latency and throughput has profound implications in Big Data research. While traditional parallel and distributed databases mostly targeted at reducing data processing latency for moderately sized datasets, Big Data systems need to take ownership costs and energy consumption into consideration. Using large quantities of small processors to achieve similar throughputs while reducing energy footprint is becoming an increasingly important topic in Big Data research. Works on using low-power ASICs [12] and FPGAs [13] and power-efficient GPUs [14], Intel MICs [15] and SoCs [16] to process Big Data have been reported in the past few years with exciting results. Several previous works on evaluating standalone ARM-based systems for relational data query processing [17][16] have shown that while these low profile systems are excellent for power-efficient computing under low utilization, they may not necessarily lead to significant energy saving under high utilization. However, we are not aware of previous works on evaluating the performance of a cluster of GPU-equipped SoCs for Big Data applications. A very recent work on evaluating Hadoop on a cluster of Exynos 5410 SoCs for relational queries using TPC-C and TPC-H benchmarks is reported in [16]. An Exynos 5410 SoC consists of 4 Cortex-A7 (little) and 4 Cortex-A15 (big) CPU cores; however, its GPU is incapable of general computing and is left unutilized. While both TK1 and Exynos 5410 include 4 Cortex-A15 CPU cores, TK1 has 192 Nvidia CUDA cores that are capable of general computing in a way the same as desktop/server grade GPUs, which makes TK1 much more powerful than Exynos 5410. Our work is orthogonal to [16] in the sense that we target at spatial data processing which is both data intensive and computing intensive; and we compare with Spark instead of Hadoop.

Since Nvidia Tegra K1 (TK1) SoC boards became available in 2014, its high computing power (up to 326 Gflops from GPU alone) and low cost (currently $192) have made it attractive in several mobile application domains, including real time image processing, computer vision and robotics. Our interests in TK1, in this study, however, are not on the absolute performance, which is still weaker than a regular CPU or a low-end discrete GPU on a desktop/server. Instead, we plan to use TK1 as down-scaled computing node to build a

low-cost and low profile cluster with high network bandwidth and high storage performance (by attaching SSDs) using commodity peripherals for Big Data research. Different from Raspberry Pi[15] SoCs which either do not provide networking capabilities or only provide 10Mbps or 100 Mbps through a USB interface, TK1 uses standard 1Gbps network interface. According to [16], data communication in Exynos 5410 also uses USB which limits its effective network bandwidth to 200-300 Mbps. The high network bandwidth to compute ratio provided by TK1, in addition to supporting SSDs through standard SATA interface, makes it ideal for our purpose. While it is beyond the scope of this study to formally model the mapping between a down-scaled TK1-based cluster and a regular cluster equipped with desktop/server grade GPUs, we hope the empirical results in a domain-specific application (spatial join query processing in Big Spatial data) can provide some insights on the feasibility and effectiveness of the proposed scaled-down based approach for Big Data research.

As a proof-of-concept, we have developed a small GPU cluster using four TK1 boards running Ubuntu Linux 14.04 with pre-configured drivers. Each TK1 board is equipped with both a multi-core CPU (4 ARM A15 cores at 2.32 GHZ) and a GPU (192 CUDA cores at 850 MHZ). As the onboard eMMC storage only has a capacity of 16GB, which is insufficient for Big Data applications, we have attached a Kingston 120GB SSDNow V300 drive to each board through a SATA 6.0 gbps interface, with half capacity used as a disk and half of the capacity used as virtual memory. The SSD drive, although inexpensive ($60), provides a read and write speed up to 450MB/s. Adding the SSD drive brings the total cost of a computing node to around $250. Adding a 24-port Buffalo Gigabit switch ($60), the 4-node tiny GPU cluster thus costs slightly over $1000, which is still lower than many desktop computers even without a discrete GPU, yet is still capable of achieving >1TFlops computing power. While the tiny GPU cluster can be used for general purpose, we are currently focusing on processing large-scale spatial data. In this study, it is not our intension to develop algorithms and implementations that are specifically designed for SoC boards, which are left for future work. Instead, we install both open source software and our in-house spatial data processing modules on the tiny GPU cluster, run experiments that we have tested for regular multi-core CPU and GPU clusters and compare their performance. We analyze the experiment results and discuss several architectural design issues for big spatial data processing on multi-core CPU and GPU equipped clusters.

III. The Tiny GPU Cluster for Distributed Spatial Join Query Processing

The architecture of the proposed tiny GPU cluster is illustrated in Fig. 1, where relevant software and parallel programming tools used in our C/C++ based implementations are also shown. Due to budget constraint, the cluster currently has only 4 nodes but the switch allows up to 24 distributed nodes. Within a node (TK1 board), each of the four ARM A15 cores has a private L1 cache and a shared L2 cache. The GPU is based on Nvidia Kepler microarchitecture and has 192 CUDA cores in a Streaming Multiprocessor (SM). Conceptually, the TK1 GPU has 1/14 of the computing power of GTX TITAN[16] which has 14 SMs. We note that according to elinux website, a TK1 board consumes ~11W power when performing computing intensive image processing tasks[17] while consumes 1.6-4.7W when running a disk-intensive search command[18]. The low energy footprint of TK1 also makes it interesting to use the tiny GPU cluster for energy efficiency related research in Big Data applications, which is left for our future work.

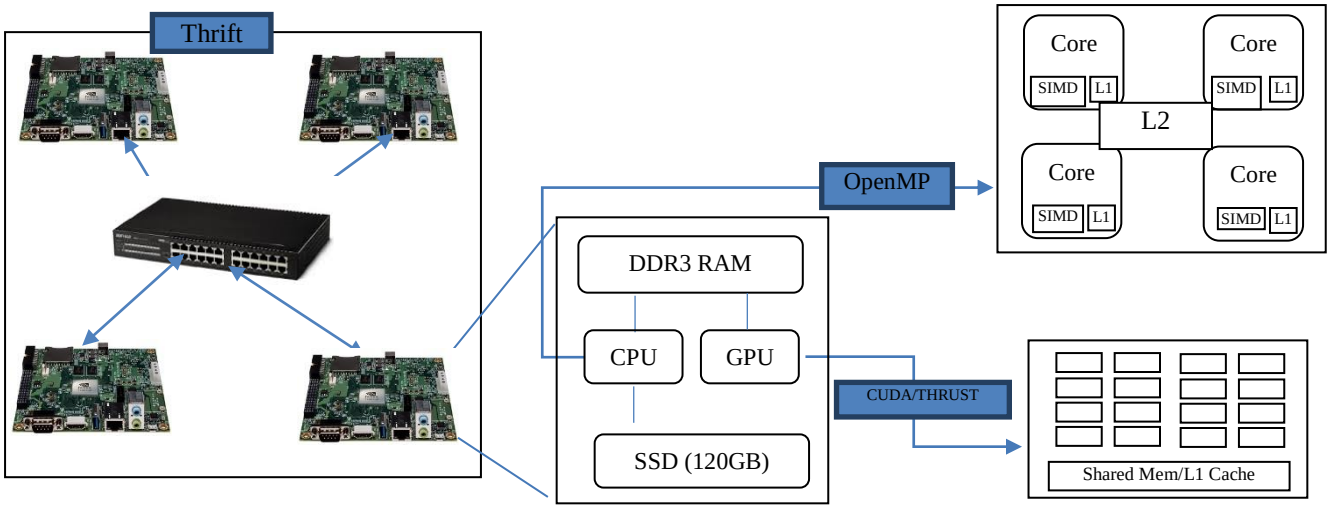

Fig. 1 Hardware Configuration and Software Stacks of a 4-node Tiny GPU Cluster

The LDE engine [8] uses Apache Thrift to define data types and service interfaces for data communication across the network. Our micro-benchmark has confirmed that the realized network bandwidth for point-to-point data communication in the tiny ("Wimpy") cluster can be close to 100 MB/s, which is the same as a regular "Brawny" cluster. For our C/C++ based implementations, we program the four

ARM cores using standard C/C++ with OpenMP [19] and we program the GPU using CUDA together with the Thrust parallel library[20] which is also part of CUDA SDK. While our parallel spatial join processing techniques and the LDE engine are developed for "Brawny" clusters, porting to the "Wimpy" cluster is rather smooth as the operating system (i.e., Ubuntu Linux) hides away the differences among ARM and X86/64 CPUs.

Porting SpatialSpark to the tiny cluster is virtually effortless, as Java Virtual Machine (JVM) and Hadoop/Spark platform provides two additional layers of portability. The only issue is that, as a 32-bit CPU, ARM A15 supports only 32-bit Java SDK which does not allow use large memory, even though we have allocated 60GB SSD as virtual memory. While this issue has less effect on C/C++ based LDE engine, which has a small memory footprint (by design), Spark requires significant amount of memory in runtimes and can only accommodate small datasets with the remaining memory. As the successor of Tegra K1, which has been announced by Nvidia as Tegra X1 [21], has adopted a 64-bit ARM CPU architecture, is likely to solve the issue. Nevertheless, high infrastructure overheads of Hadoop and Spark makes it unlikely for SoC-based clusters to achieve competitive performance even after taking the scaling factor into consideration. The experiment results to be presented in Section III supports our observation. As such, we will focus on evaluating the performance of the LDE engine due to its low infrastructure overheads. While we refer to our technical report for the design and implementation details of the LDE engine and its application to distributed spatial query processing [8], for the sake of completeness, we next provide a brief introduction from a Big Spatial Data application perspective. We use one of the experiments in Section IV for illustration purpose.

Given a large taxi trip data (point dataset) and a census block data (polygon dataset), the point-in-polygon test based spatial join can be used as an example to introduce spatial data management as a concrete Big Spatial Data application. A taxi trip typically has a GPS-recorded pickup location (Origin) and a drop-off location (Destination), together with many other attributes, e.g., timestamps and fare amounts [5]. Counting the numbers of pickup/drop-off locations by census blocks and providing spatially aggregated statistics using different temporal hierarchies (e.g., hourly, daily, weekdays/weekends and peak/off-peak), can be very useful to help better transportation and city planning [5]. To align points to census blocks, a technique in spatial data management known as spatial join is required. Our LDE engine adopts a "left-partition and right-broadcast" strategy to partition point data across distributed computing nodes and broadcast polygon data to all participating distributed computing nodes for distributed query execution [8]. The LDE engine accesses HDFS to retrieve both raw data and index files in binary format, supports asynchronous network communication, asynchronous disk I/O and asynchronous computing, and support using native parallel programming tools for local processing. The features make the LDE engine efficient in processing large-scale data, when compared with our previous works on extending Impala for spatial data [6][7][8].

IV. Experiments and Results

We have performed experiments using two real world large geospatial datasets to test the performance and scalability of the tiny GPU cluster for point-in-polygon test based spatial join. The first experiment uses the taxi trip and census data (termed as *taxi-nycb* experiments) where the pickup locations of approximate 170 million taxi trips in NYC in 2013 are used as the point dataset and the 2000 census blocks in the same region with 38,794 polygons are used as the point dataset. As the average number of points per polygon is only around 9, the experiment is more data intensive. Our second experiment to spatially joining a subset of global species occurrence locations with World Wide Fund (WWF) global ecological regions. The number of points is approximately 10 million and the number of polygons is 14,458 and we term the experiment as *g10m-wwf*. As the average number of vertices of the polygons in the WWF polygon dataset is about 279, the experiment is much more computing intensive.

In addition to performing the two experiments using 1, 2 and 4 nodes of the tiny GPU cluster, similar to the experiments on regular GPU clusters as reported in [7][8], we have also measured the runtimes in a standalone setting where the distributed infrastructure is removed and computing is iteratively performed on a single node. The results on multi-core CPUs (labeled as LDE-MC) and GPUs (labeled as LDE-GPU) are listed in Table I. For the first experiment (*taxi-nycb*), we also report runtimes of SpatialSpark for comparison purposes. We have not reported the performance of SpatialSpark for the second experiment (*g10m-wwf*.) as it cannot complete within a reasonable time.

TABLE I. RUNTIMES OF OF THE TINY GPU CLUSTER UNDER MULTIPLE SETTINGS FOR TAXI-NYCB AND G10M-WWF EXPERIMENTS

Experiment	Setting	***standalone***	***1-node***	***2-node***	***4-node***
taxi-nycb	LDE-MC	18.6	27.1	15.0	11.7
	LDE-GPU	18.5	26.3	17.7	10.2
	SpatialSpark	-	179.3	95.0	70.5
g10m-wwf	LDE-MC	1029.5	1290.2	653.6	412.9
	LDE-GPU	941.9	765.9	568.6	309.7

From Table I, we can see that when the number of nodes in the cluster is increased from 1 to 4, the speedup is between 2.3X to 3.1X among all the experiment settings, which is reasonable. While LDE-GPU still performs better than LDE-MC in general (with only one exception, i.e., LDE-MC using 2 nodes) on the tiny cluster, the speedups are typically lower than those using a regular cluster as reported in [7][8]. This can be due to the reason that CPU and GPU in a TK1 board access the same DDR3 memory. Compared with GDDR5 memory on desktop/server grade GPUs with 384-bit memory bandwidth, the memory access latency is high and the bandwidth is much lower for TK1 GPU. Nevertheless, both LDE-MC and LDE-GPU have achieved significant speedups over SpatialSpark (5.4X to 6.9X) in the *taxi-nycb* experiment. The high efficiency may be due to several differences among

them: language (C/C++ vs. Scala/Java), platform (LDE vs. Spark) and implementation efficiency of point-in-polygon test. While we leave a more detailed analysis to future work, we argue that a lightweight distributed execution engine for the tiny cluster is advantageous. While we are able to successfully set up Hadoop and Spark on the tiny cluster, we have encountered several major technical challenges to port our ISP-based systems [6][7], which extend Impala, to the tiny GPU cluster due to the complexities of Impala and its various required third party software packages. Even though we have run SpatialSpark successfully on the tiny cluster, the performance is less preferable when compared with the LDE engine that is lightweight with less than 1,000 lines of code [8].

It is also interesting to compare the performance of the tiny GPU cluster with a regular GPU cluster. We have increased the number of species occurrence records in the *g10m-wwf* experiment from approximately 10 million to approximately 50 million and run the experiment on the tiny GPU cluster with 4 nodes. The new experiment (termed as *g50m-wwf*) is thus more computing intensive and takes longer on the tiny cluster. The performance results of the experiment on an Amazon EC2 GPU cluster have been reported in [8] and both results are tableted in Table II. For comparison purpose, the performance on a high-end workstation in the standalone setting is also listed in Table II.

TABLE II. RUNTIMES UNDER DIFFERENT HARDWARE CONFIGURATIONS FOR THE G50M-WWF EXPERIMENT

	TK1-Standalone	***TK1-4 Node***	***Workstation-Standalone***	***EC2-4 Node***
CPU Spec. (per node)	ARM A15 2.34 GHZ 4 Cores 2 GB DDR3		Intel SB 2.6 GHZ 16 cores 128 GB DDR3	Intel SB 2.6 GHZ 8 cores (virtual) 15 GB DDR3
GPU Spec. (per node)	192 Cores 2GB DDR3		2,688 cores 6 GB GDDR5	1,536 cores 4 GB GDDR5
Runtime (s) –MC	4478	1908	350	334
Runtime (s) –GPU	4199	1523	174	105

From Table II we can see that, when comparing the runtimes of the CPU implementation in standalone setting, the workstation with dual eight-core Intel Sandy Bridge (SB) CPUs is about 12.8X faster than TK1 with 4 ARM CPU cores. Given an Intel CPU with 8 cores consumes 95W while an ARM A15 CPU with 4 cores consumes 10W, the workstation consumes 95*2/10=19X more power. This suggests that the ARM CPU on TK1 can potentially achieve 95*2/10/12.8=1.48X more energy efficiency in the standalone setting. On the other hand, when only energy consumed by CPUs (at maximum level) is taken into consideration, the EC2 4-node cluster consumes 95/10=9.5X more energy but is 1908/334=5.7X faster, which may suggest the TK1 4-node cluster can be 9.5/5.7=1.67X more energy efficient.

While the GPU model in the Amazon EC2 cluster is not available to us (which could be virtualized GPU), the workstation is known to be equipped with GTX Titan. In standalone setting, the GPU implementation on the workstation is 24X faster than TK1. Given that GTX Titan has 14X more CUDA cores, there is no evidence in this experiment showing that the TK1 cluster is more performant when only the number of GPU cores is taken into consideration. The workstation can be 24/14=1.7X more efficient than TK1. A similar trend can be observed on the cluster performance with 4 nodes: the EC2 4-node cluster has 8X more CUDA cores while runs 14X faster which may suggest that the EC2 4-node cluster is 1.75X more efficient than the TK1 4-node cluster.

While it is tempting to draw a conclusion that TK1 CPU has a better energy efficiency than that of GPU for the experiment in both the standalone setting and the 4-node cluster setting by assuming that all CUDA cores consumes the same amount of energy, we argue that more research is needed. The runtimes of GPU implementations do not necessarily translate into energy consumption linearly as energy consumed by CPUs, memory, disk and network cannot be easily incorporated in the simple energy model. Furthermore, it is likely that GPU may become idle while waiting for resources during the process. We leave more in-depth investigation for future work.

V. CONCLUSIONS AND FUTURE WORK

Motivated by the increasing gap between the computing power of GPU-equipped clusters and network bandwidth and disk I/O throughput, we propose to develop a low-cost prototype research cluster made of Nvidia TK1 SoC boards that can be interconnected with standard 1Gbps network to facilitate Big Data research. By porting our previous works on distributed spatial query processing techniques, include SpatialSpark and the LDE engine, we evaluate the performance of the tiny GPU cluster for spatial join query processing on large-scale geospatial data. Experiments on point-in-polygon test based spatial join using two real world applications with tens to hundreds of millions of points and tens of thousands of polygons have demonstrated the efficiency of the LDE engine when compared with SpatialSpark. Preliminary analysis on the scaling effect between the tiny cluster and a regular Amazon EC2 cluster using a simplified model seems to suggest that the ARM CPU of the TK1 board is likely to achieve better energy efficiency while the Nvidia GPU of the TK1 board is less performant when compared with desktop/server grade GPUs, in both the standalone setting and the 4-node cluster setting for the two particular application.

For future work, first of all, we would like to develop a formal method to model the scaling effect between SoC-based clusters and regular clusters, not only including processors but also memory, disk and network components. Second, while it is beyond our scope to develop benchmarks for all types of data, we would like to evaluate the performance of SpatialSpark and the LDE engine using more real world geospatial datasets and applications, e.g., distance and nearest neighbor based spatial joins. We hope our work in processing large-scale geospatial data, together with the mainstream Big Data research on relational and graph data,

can play a complementary role in understanding the interplays between hardware and software and develop novel techniques for more efficient and scalable Big Data applications.

ACKNOWLEDGEMENT

This work is supported through NSF Grants IIS-1302423 and IIS-1302439.

VI. REFERENCES

[1] S. Jha, J. Qiu, A. Luckow, P. Mantha and G. Fox, "A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures," in Proceedings of *IEEE International Congress on Big Data (BigData Congress)*, Anchorage, AK , 2014.

[2] X. Lu, F. Liang, B. Wang, L. Zha and Z. Xu, "DataMPI: Extending MPI to Hadoop-Like Big Data Computing," in Proceedngs of *IEEE 28th International Parallel and Distributed Processing Symposium*, Phoenix, AZ , 2014.

[3] Y. Wang, ong, Y. Jiao, C. Xu, X. Li, T. Wang, X. Que, C. Cira, B. Wang, Z. Liu, B. Bailey and W. Yu, "Assessing the Performance Impact of High-Speed Interconnects on MapReduce," in *Specifying Big Data Benchmarks*, Berlin Heidelberg, Springer , 2014, pp. 148-163.

[4] N. S. Islam, M. W. Rahman, J. Jose, Rajach, R. rasekar, H. Wang, H. Subramoni, C. Murthy, P and D. K. a, "High Performance RDMA-based Design of HDFS over InfiniBand," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, Salt Lake City, Utah, 2012.

[5] J. Zhang, S. You and L. Gruenwald, "Parallel Online Spatial and Temporal Aggregations on Multi-core CPUs and Many-Core GPUs," *Information Systems*, vol. 44, p. 134–154, 2014.

[6] S. You, J. Zhang and L. Gruenwald, "Large-Scale Spatial Join Query Processing in Cloud," in *Proceedings of International Workshop on Cloud Data Management (CloudDM'15)*, Seoul, KOREA, 2015 (to appear).

[7] S. You, J. Zhang and L. Gruenwald, "Scalable and Efficient Spatial Data Management on Multi-Core CPU and GPU Clusters: A Preliminary Implementation based on Impala," *in Proceedings of International Workshop on Big Data Management on Emerging Hardware (HardBD'15)*, Seoul, KOREA , 2015 (to appear).

[8] J. Zhang, S. You and L. Gruenwald, "A Lightweight Distributed Execution Engine for Large-Scale Spatial Join Query Processing," technical report, online at http://www-cs.engr.ccny.cuny.edu/~jzhang/papers/lde_spatial_tr.pdf, 2015.

[9] Y. Wang, C. Xu, X. Li and W. Yu, "JVM-Bypass for Efficient Hadoop Shuffling," in Proceedings of *IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*, Boston, MA , 2013.

[10] W. Yu, Y. Wang and X. Que, "Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 602-611, 2014.

[11] Y. Wang, R. Goldstone, W. Yu and T. Wang, "Characterization and Optimization of Memory-Resident MapReduce on HPC Systems," in Proceedings of *IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*, Phonix, AZ, 2014.

[12] L. Wu, Lottarini, rea, T. K. Paine, M. A. Kim and K. A. Ross, "Q100: The Architecture and Design of a Database Processing Unit," in Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14), *2014*, Salt Lake City, Utah, USA.

[13] O. Arnold, S. Haas, G. Fettweis, B. Schlegel, T. Kissinger and W. Lehner, "An Application-specific Instruction Set for Accelerating Set-oriented Database Primitives," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*, Snowbird, Utah, USA, 2014.

[14] H. Wu, G. Diamos, T. Sheard, M. Aref, S. Baxter, Garl, M. and S. Yalamanchili, "Red Fox: An Execution Environment for Relational Query Processing on GPUs," in *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO'14)*, Orlando, FL, USA, 2014.

[15] M. Lu, L. Zhang, H. P. Huynh, Z. Ong, Y. Liang, B. He, R. Goh and R. Huynh, "Optimizing the MapReduce framework on Intel Xeon Phi coprocessor," in Proceedings of *IEEE International Conference on Big Data (BigData'13)*, 2013.

[16] D. Loghin, B. M. Tudor, H. Zhang, B. C. Ooi and Y. M. Teo, "A Performance Study of Big Data on Small Nodes," *PVLDB*, vol. 8, no. 7, pp. 762-773, 2015.

[17] T. Muhlbauer, W. Rodiger, R. Seilbeck, A. Kemper and T. Neumann, "Heterogeneity-conscious parallel query execution: getting a better mileage while driving faster!," in *Proceedings of the Tenth International Workshop on Data Management on New Hardware (DaMoN'14)*, 2014.

L. Wu, R. J. Barker, M. A. Kim and K. A. Ross, "Navigating Big Data with High-throughput, Energy-efficient Data Partitioning," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*, 2013.

---

[1] https://spark.apache.org/
[2] https://github.com/cloudera/impala
[3] http://aws.amazon.com/ec2/instance-types/
[4] http://en.wikipedia.org/wiki/Spatial_database
[5] http://en.wikipedia.org/wiki/Geographic_information_system
[6] http://en.wikipedia.org/wiki/InfiniBand
[7] https://developer.nvidia.com/jetson-tk1
[8] http://www.nvidia.com/object/cuda_home_new.html
[9] http://netty.io/
[10] http://en.wikipedia.org/wiki/Message_Passing_Interface
[11] http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[12] http://en.wikipedia.org/wiki/Remote_direct_memory_access
[13] http://en.wikipedia.org/wiki/Java_Native_Interface
[14] http://lustre.org/
[15] http://www.raspberrypi.org/
[16] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan
[17] http://elinux.org/Jetson/Computer_Vision_Performance
[18] http://elinux.org/Jetson/Jetson_TK1_Power
[19] http://openmp.org/wp/
[20] https://thrust.github.io/
[21] http://www.nvidia.com/object/tegra-x1-processor.html