When Apache Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration

Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei and Peng Wei
{*ytchen, cong, zhenman, jielei and peng.wei.prc*}*@cs.ucla.edu*
University of California, Los Angeles

Abstract

FPGA-enabled datacenters have shown great potential for providing performance and energy efficiency improvement. In this paper we aim to answer one key question: *how can we efficiently integrate FPGAs into state-of-the-art big-data computing frameworks like Apache Spark?* To provide a generalized methodology and insights for efficient integration, we conduct an in-depth analysis of challenges at single-thread, single-node multi-thread, and multi-node levels, and propose solutions including batch processing and the FPGA-as-a-Service framework to address them. With a step-by-step case study for the next-generation DNA sequencing application, we demonstrate how a straightforward integration with 1,000x slowdown can be tuned into an efficient integration with 2.6x overall system speedup and 2.4x energy efficiency improvement.

1 Introduction

Nowadays, power and energy efficiency of general-purpose processors have become two of the primary constraints that limit the performance scaling of conventional datacenters. Harnessing FPGA-based heterogeneous platforms, which provide low power, high energy efficiency and reprogrammability, is considered one of the most promising approaches to yield continued performance and energy efficiency improvement. For example, Microsoft has deployed FPGAs into its datacenters to accelerate the ranking stage of the Bing search engine with almost 2x throughput improvement while consuming only 10% more power [17]. IBM has also deployed FPGAs in its data engine for large and fast-growing NoSQL data stores [4]. In addition, Intel, with acquisition of Altera, is providing QPI-based CPU-FPGA platforms for datacenters [10]. Predictably, there will be many FPGA-enabled datacenters in the near future.

With the emerging FPGA-enabled datacenter trend, one key question is: *how can we efficiently integrate FPGAs into state-of-the-art big-data computing frameworks like Apache Spark [20]?* According to our case study, a straightforward FPGA integration can actually lead to a slowdown by 1,000x compared to a CPU-only cluster. Our goal is to provide generalized insights for efficient integration of FPGA accelerators into the Spark MapReduce framework [20], and turn the slowdown back to performance and energy efficiency improvement.

Our approach is to conduct an in-depth case study for the acceleration of an important and representative application: next-generation DNA sequencing [18]. We choose this application for the following two reasons. First, it is an important application that is transitioning into clinical use where time is a matter of life and death. Moreover, the FPGA accelerator for this application represents a category of fine-grained accelerators that impose further challenges to the integration. Unlike conventional coarse-grained accelerators, these fine-grained accelerators execute for a very short time (e.g., a microsecond or so) but will be invoked many (e.g., hundreds of millions) times, and thus a straightforward offloading of the CPU computation onto the FPGA board could significantly degrade the overall performance due to the overwhelming JVM-FPGA communication overhead (e.g., a few milliseconds for data to be transferred from JVM to native machine and then to FPGA).

In summary, this paper makes the following contributions.

1. Methodology and insights for efficient integration of FPGAs into big-data computing frameworks like Spark, including what challenges are expected at single-thread (Section 4.1), single-node multi-thread (Section 4.2), and multi-node levels (Section 4.3), as well as how to address them.
2. Design and deployment of an FPGA-enabled Spark cluster that features batch processing to alleviate JVM-FPGA communication overhead and the FPGA-as-a-Service (FaaS) framework to efficiently share FPGAs among multiple CPU threads, achieving 2.6x better performance than a CPU-only cluster for the emerging DNA sequencing application, while consuming only 8% more power per server.

2 Background and Related Work

There is an increasing trend to integrate FPGA accelerators into modern datacenters. For example, Microsoft has developed a customized FPGA board, Catapult, and placed it into each server to accelerate the ranking stage of the Bing search engine in a 1,632-node cluster [17]. With a key focus on discussing the robust design of the large-scale system architecture, this publication did not reveal many details of the programming framework. Moreover, IBM has proposed the Coherent Accelerator Processor Interface (CAPI) to connect a PCIe-based

FPGA board to a POWER8 processor, and integrated such FPGAs into its in-memory data structure store Redis to accelerate its Data Engine for NoSQL [4]. In this paper we aim to provide a more generalized methodology and insight for efficient integration of FPGA accelerators into state-of-the-art big-data computing frameworks like Spark, and therefore stimulate more innovations in this very hot area.

Meanwhile, there are also some efforts that integrate GPU accelerators into Hadoop and Spark. For example, in [8], Grossman et al. proposed an automated flow to generate OpenCL kernels for Hadoop programs in a GPU-equipped cluster. In [14], Li et al. integrated GPU accelerators with Spark for deep learning algorithms. While these approaches usually target the integration of coarse-grained accelerators, we mainly focus on the integration of fine-grained FPGA accelerators, which introduces more challenges, like efficient communication and sharing as discussed in Section 4.1 and Section 4.2.

3 Cluster Scale Acceleration: Case Study

3.1 Next-Generation DNA Sequencing

Next-generation sequencing results from a combination of chemical engineering and computer science innovations. To sequence a human's entire genome, a number of copies of the individual's genome are fragmented into small pieces, called *reads*, and the sequencers determine the order of nucleotides for each read. The sequenced reads are stored as ASCII strings (roughly 100 characters each), and aligned to specific locations of a reference genome (a string of 3 billion characters) to be assembled into an entire DNA sequence.

Generally, a sequencing instance processes billions of reads, and each read is independently sequenced and aligned. This billion-degree parallelism makes it a good candidate for cluster scale acceleration. While single-machine software tools, such as Burrows-Wheeler Aligner (BWA) [13], Bowtie [12] and Genome Analysis Toolkit (GATK) [15] are still widely used for read alignment and successive data analysis, a few cluster scale tools have been proposed to serve as alternatives. In [6], Chen et al. proposed CS-BWAMEM, a Spark-based MapReduce implementation for mapping the short reads onto the reference genome. In [16], Massie et al. proposed ADAM, another Spark-based implementation, which provides a set of formats, APIs and tools for data analysis on aligned reads.

Conceptually, the sequencing algorithm consists of two phases, *seeding* and *extending*. In the seeding phase, a read uses its substrings of various lengths, called *seeds*, to find candidate alignment positions on the reference genome. In the extending phase, each seed is extended leftward and/or rightward to both ends of the read via a two-dimension dynamic programming algorithm, the Smith-Waterman (S-W) algorithm [19]. In this paper we focus on the extending phase of CS-BWAMEM (since it is more time-consuming) and present the integration process of CS-BWAMEM and a S-W FPGA accelerator.

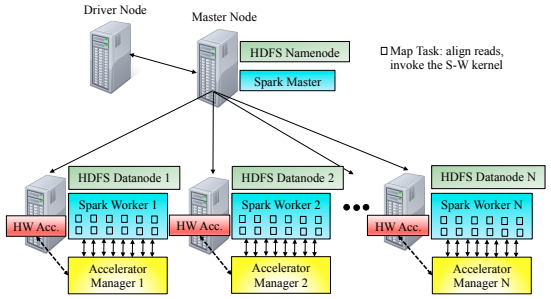

Figure 1: An overview of the Spark-FPGA cluster

3.2 FPGA Acceleration for S-W Algorithm

FPGA acceleration for the S-W algorithm has attracted great attention in the past. Various approaches have been proposed and implemented and achieves over 100x speedup compared to CPU and even GPU based solutions [21][3][5][2]. While the proposed accelerators show a great potential for accelerating the S-W computational kernel, FPGA researchers did not pay enough attention to the system-wide integration of the accelerators. In this paper we focus on the integration of the S-W accelerator in [5] into the Spark-based CS-BWAMEM, where the FPGA accelerator achieves around 120x and 10.5x kernel-level speedup over the single-thread and 16-thread CPU in our experimental system.

3.3 Experimental Setup

Our experimental system comprises a cluster of 1 master node and 6 worker nodes, as shown in Figure 1. Except for the master node of the Spark framework, all Spark's worker nodes are equipped with a PCIe-attached Alpha Data ADM-PCIE-7V3 FPGA board [1]. Table 1 lists the detailed configuration of each server. Currently, we have only six FPGA boards available, which limits the cluster size. Nevertheless, it is sufficient to demonstrate our integration methodology and insights, which can be easily applied to larger clusters. We are planning to incorporate more FPGA boards into our cluster in the future.

Table 1: Experimental setup

Host CPU	two 6-core Xeon E5-2620v3@2.40GHz
Host Memory	48GB DDR3-1600
FPGA Fabric	Xilinx Virtex 7@200MHz
CPU $\leftrightarrow$ FPGA	PCIe Gen3 x8, 8GB/s as advertised
FPGA Device Memory	16GB DDR3-1600
Development Environment	SDAccel 2015.1.5

We use Spark 1.5.1 as our cluster computing framework and HDFS 2.5.2 as our underlying distributed file system, and run CS-BWAMEM 0.2.2 on top of them. As illustrated in Figure 1, each read is aligned by a CS-BWAMEM's map function that invokes the S-W kernel. Our test cases are derived from the genome sample of

a human with breast cancer (HCC1954 [7]). The sample contains almost 1 billion reads, each with 101 nucleotides (denoted by a 101-character ASCII string). The performance of DNA sequencing applications is often measured by the number of reads aligned in a unit of time. In this paper we use the notation of "kilo reads per second (KRPS)".

For convenience, we will denote the original CS-BWAMEM program as CS-BWAMEM/CPU, and the CS-BWAMEM program with the S-W accelerator as CS-BWAMEM/FPGA in the rest of this paper. CS-BWAMEM will be also used in the scenarios where both CS-BWAMEM/CPU and CS-BWAMEM/FPGA fit.

4 Challenges and Solutions

4.1 Harnessing FPGA in JVM

Spark programs are mainly written in Java and/or Scala, and run on JVMs. FPGA accelerators are typically manipulated by C/C++ programs, and JVMs do not support the use of FPGAs by default. Therefore, the first step of Spark-FPGA integration at the single-thread level is to bridge the gap between Java/Scala and C/C++.

While the Java Native Interface (JNI) serves as a standard tool to address this issue, it does not always deliver an efficient solution. In the single-thread scenario, we compare the performance of CS-BWAMEM/CPU and CS-BWAMEM/FPGA, and find that CS-BWAMEM/CPU achieves 2.1 KRPS (kilo reads per second) while CS-BWAMEM/FPGA, with the straightforward Spark-FPGA integration, reaches merely 1.6 RPS. In other words, the straightforward integration does not fulfill the 120x speedup, but instead decreases the overall performance by three orders of magnitude.

After an in-depth analysis, we find that the main reason for the performance degradation is the tremendous JVM-FPGA communication overhead aggregated through all the invocations of the S-W accelerator. To be specific, one read produces 24 S-W invocations (either software or hardware implementation) on average, and it takes about 480μs for the software to process them in JVM. That is, each S-W invocation of the software version should cost no more than 20μs on average. Meanwhile, a complete routine of a S-W accelerator invocation involves: 1) data copy between a JVM and a native machine, 2) DMA transfer between a native machine and an FPGA board though PCIe, and 3) computation on the FPGA board. The communication process, including 1) and 2), costs over 25ms per invocation. That is, even if an accelerator could reduce the computation time of the S-W kernel down to 0, the communication overhead would degrade the performance by 1000x.

The tremendous JVM-FPGA communication overhead has to be alleviated to make the Spark-FPGA integration work efficiently. In CS-BWAMEM/CPU, each

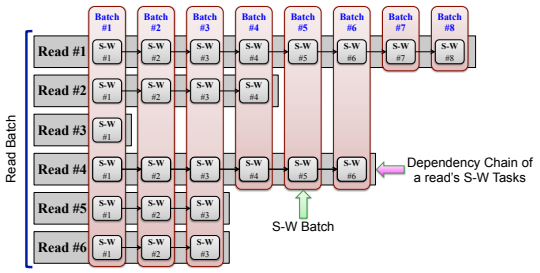

Figure 2: Batch processing in CS-BWAMEM

S-W task has only 1-2KB input data and 20B output data. These small payloads result in an extremely low communication bandwidth utilization of both DRAM (from a JVM to a native machine) and PCIe (from a native machine to an FPGA). This phenomenon motivates our approach of batching a group of reads together and offloading them to the FPGA board at a time to improve the bandwidth utilization.

To make batch processing work, a fundamental condition is that there should be adequate independent tasks to process as a whole. A Spark MapReduce program inherently offers a massive degree of parallelism. All map function calls in a map stage are completely independent of each other. Therefore, it is both necessary and feasible to conduct batch processing for CS-BWAMEM. To be specific, we merge a certain number of CS-BWAMEM/FPGA's map tasks (derived from the straightforward integration) into a new map function, and conduct a series of code transformations to batch the S-W kernel invocations from different map tasks together.

However, there is a delicate issue in CS-BWAMEM that imposes challenges in the code transformation for batch processing, which is illustrated in Figure 2. First, a read generates N leftward/rightward extending tasks, indicating that a map function of CS-BWAMEM (before the code transformation for batch processing) needs to process N S-W tasks (a row in Figure 2), where N is highly varied for different reads. Moreover, all these S-W tasks generated in the same read are chain-dependent (a row in Figure 2) and thus cannot be batched together. Therefore, a batched map function has to consider multiple reads (rows) as a group and produce multiple S-W batches (each column a batch) for this group. To better hide the communication, only S-W batches (within reads) with batch size no less than a threshold (64 in our experiments) would be offloaded to the FPGA accelerator. All other smaller S-W batches are processed on CPU.

Figure 3 shows the performance of processing a set of reads in CS-BWAMEM/FPGA with different batch sizes (the number of reads in a batch), as well as the performance of CS-BWAMEM/CPU. We can see that the FPGA integration starts to outperform the CPU-only version when the read batch size reaches 16k. The performance continues to increase until the program runs

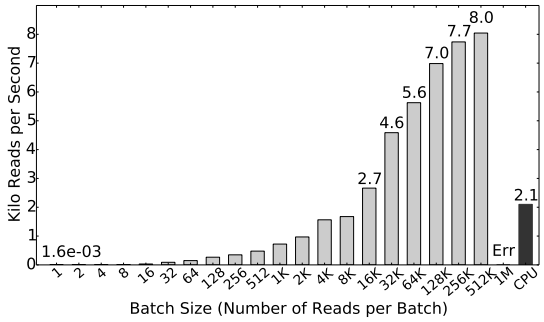

Figure 3: Performance under different read batch sizes

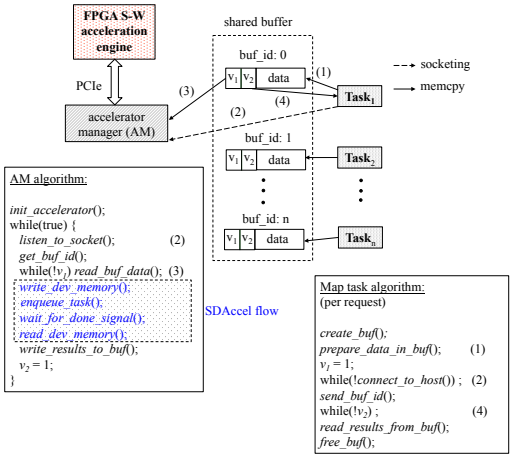

Figure 4: FPGA as a Service (FaaS) framework

out of memory, where the batch size exceeds 512k. With the batch size of 512k, CS-BWAMEM/FPGA achieves around 8 KRPS and is around 4x faster than CS-BWAMEM/CPU in the single-thread case.

4.2 Sharing FPGAs Among Threads

Due to the high performance of FPGA accelerators, offloading a single-thread CPU workload onto the FPGA usually makes the FPGA underutilized, which leaves opportunities for FPGA accelerators to be shared by multiple threads in a single node. The major challenge is how to efficiently manage the FPGA accelerator resources among multiple CPU threads. To tackle this challenge, we propose an FPGA-as-a-Service (FaaS) framework and implement the FPGA management in a node-level accelerator manager.

The FaaS framework abstracts the FPGA accelerator and its management software on the CPU (called Accelerator Manager (AM)) as a *server*, and treats each CPU thread as a *client*. Client threads communicate with AM via a hybrid of JNI and network sockets. Different client threads send requests independently to the AM to accelerate S-W batches, and the AM processes the requests in a first-come-first-serve way. Figure 4 describes the functionality and the detailed implementation of the FaaS framework.

In the single-thread version, CS-BWAMEM/FPGA's map functions have been modified into batched map

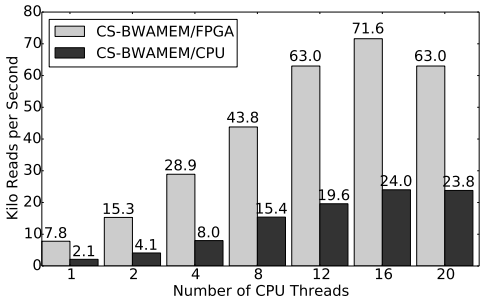

Figure 5: Performance under different number of threads

functions. The S-W tasks from these map functions have also been reorganized into a series of S-W batches, which are sent through JNI to the native library that manipulates the FPGA accelerator. The FaaS framework extends the native library into AM, and extends the communication mechanism between the batched map function and AM as follows.

1. When a batched map function in a CPU thread needs to use the FPGA accelerator, it will first allocate a shared memory buffer and then send the input data from JVM to this buffer through JNI.
2. The batched map function sends a request to AM through a socket to use the accelerator. The request contains only the address of the shared buffer created in Step 1, thus generating negligible overhead.
3. If the accelerator is available, it will be locked and start to process the S-W batch; otherwise, the batched map function waits in a spin loop until it successfully gets permission to use the accelerator.
4. After the accelerator completes the current S-W batch, it will write the output data back to the shared memory buffer created in Step 1, and become available again to accept another request.

Figure 5 shows the performance comparison between CS-BWAMEM/CPU and CS-BWAMEM/FPGA with different number of CPU threads. We can see that the speedup of FPGA-equipped system over the CPU-only system slightly decreases from about 4x (single-thread) to 3x (16 hyper-threads) due largely to thread contention, but still maintains a decent speedup. The performance of CS-BWAMEM/FPGA starts to decrease when 20 hyper-threads share the FPGA board. Meanwhile, the performance of CS-BWAMEM/CPU sightly decreases at this point as well, which indicates that hyper-threading does not always help performance improvement for CS-BWAMEM. Therefore, we will use 16 hyper-threads per CPU throughout this paper since it achieves the best performance for both CS-BWAMEM/CPU and CS-BWAMEM/FPGA.

4.3 Scaling FPGAs into Cluster Scale

Finally, we address challenges when scaling FPGA integration into cluster scale. Based on our case study, when a computational kernel is inside a map function of a Spark MapReduce program, the inter-node communica-

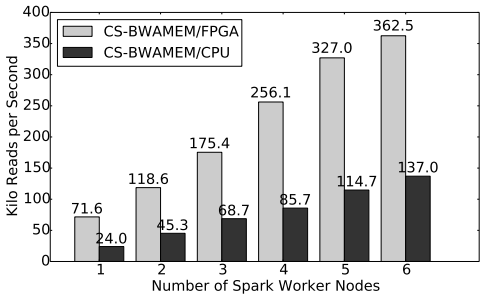

Figure 6: Performance under different number of nodes

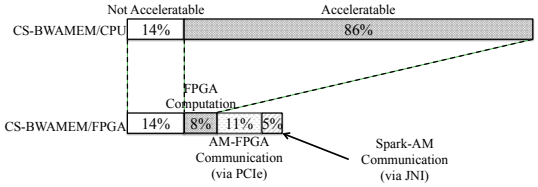

Figure 7: CS-BWAMEM execution time breakdown

tion will be completely independent of the FPGA board in each server. That is, Spark application developers who wants to harness the power of FPGA accelerators for computational kernels residing in map functions merely need to consider up to the single-node multi-thread level.

Overall Performance and Power. After overcoming all the challenges at various levels, now we have an efficient integration of CS-BWAMEM/ FPGA. As shown in Figure 6, the performance of Spark-FPGA integration scales well with one to six worker nodes, where each node runs 16 threads. Through the efficient integration of FPGA accelerators, CS-BWAMEM/FPGA improves the overall system performance of a 6-worker cluster by 2.6x, compared to CS-BWAMEM/CPU. Under the same configuration, CS-BWAMEM/FPGA consumes only 8% additional power per worker node. That is, CS-BWAMEM/FPGA achieves 2.4x energy efficiency improvement and 2.6x performance speedup. This result goes along with Microsoft's findings for the ranking stage of the Bing search engine where the performance is improved by 2x while consuming 10% more power per server. It is quite promising that FPGAs can greatly improve performance and energy efficiency in datacenters.

Analysis of Communication Overhead. To better demonstrate the effectiveness of FPGA acceleration, we present the detailed execution time breakdown (normalized to the CS-BWAMEM/CPU baseline) of our 6-worker Spark-FPGA system in Figure 7. The upper bar illustrates that the S-W accelerator targets at 86% of the overall execution time, where the rest 14% of time mainly involves Spark's task scheduling and the S-W tasks that are processed on CPU. As shown in the lower bar, the S-W accelerator reduces the acceleratable part (86%) to 8% while paying a 16% communication overhead. The communication between the Spark program and AM through JNI introduces 5% overhead, and the communication between AM and the FPGA accelerator through PCIe introduces another 11% overhead. The existence of the above communication overhead reduces the overall system speedup to 2.6x. We can see that there is still room to improve the overall performance if FPGA fabric can be brought closer to CPU so as to further reduce the CPU-FPGA communication overhead.

5 Lessons Learned and Open Discussion

This paper presented an in-depth analysis of challenges and corresponding solutions when integrating FPGA accelerators into Spark at single-thread, single-node multi-thread, and multi-node levels. Using the next-generation DNA sequencing application CS-BWAMEM and its Smith-Waterman accelerator integration as a case study, we demonstrated how we turned a 1000x slowdown of the straightforward integration into an efficient integration with 2.6x system-wide performance improvement, at the cost of consuming only 8% more power.

We summarize some lessons we learned and discuss some open topics as below.

1. By decoupling the design and implementation of the FPGA accelerator and scale-out software, it is feasible to efficiently integrate existing FPGA accelerators into MapReduce programs with affordable programming efforts. Our case study shows that an over 2x speedup and energy efficiency can be achieved with modest code transformations.
2. It is important to generalize the FaaS framework to automatically manage and share FPGA accelerators in the Spark-FPGA integration. There are also research opportunities to further extend it to a generic accelerator management framework that harnesses a variety of heterogeneous devices, such as GPUs, FPGAs and even ASICs. At UCLA, we are developing a runtime system called Blaze [9][11] that extends Spark to enable automatic FPGA and GPU accelerator sharing among threads, as well as task pipelining to alleviate JVM-FPGA communication overhead.
3. The JVM-FPGA communication overhead appears as a primary performance bottleneck in Spark-FPGA integration. Worse still, one of our future studies, DNA sequencing acceleration on a Spark-GPU-FPGA cluster, also demonstrates the same issue on the GPU side. Batch processing alleviates this overhead to some extent, but it is not always trivial to transform a given MapReduce program into a batched style. It could be greatly helpful if batching-oriented programming model extensions or (semi)-automated code transformation tools could be proposed for MapReduce.

Acknowledgments

This work was supported in part by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and by NSF/Intel Innovation Transition Grant awarded to the Center for Domain-Specific Computing.

References

[1] Alpha Data ADM-PCIE-7V3 datasheet. URL: http://www.alpha-data.com/pdfs/adm-pcie-7v3.pdf.

[2] AHMED, N., SIMA, V.-M., HOUTGAST, E., BERTELS, K., AND AL-ARS, Z. Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (Piscataway, NJ, USA, 2015), ICCAD '15, IEEE Press, pp. 240–246.

[3] ARRAM, J., TSOI, K., LUK, W., AND JIANG, P. Hardware acceleration of genetic sequence alignment. In *Reconfigurable Computing: Architectures, Tools and Applications*, P. Brisk, J. de Figueiredo Coutinho, and P. Diniz, Eds., vol. 7806 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 13–24.

[4] BRECH, B., RUBIO, J., AND HOLLINGER, M. IBM Data Engine for NoSQL - Power Systems Edition. Tech. rep., IBM Systems Group, 2015.

[5] CHEN, Y.-T., CONG, J., LEI, J., AND WEI, P. A novel high-throughput acceleration engine for read alignment. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on* (May 2015), pp. 199–202.

[6] CHEN, Y.-T., CONG, J., LI, S., PETO, M., SPELLMAN, P., WEI, P., AND ZHOU, P. CS-BWAMEM: A fast and scalable read aligner at the cloud scale for whole genome sequencing. *High Throughput Sequencing Algorithms and Applications (HITSEQ)* (2015).

[7] GAZDAR, A. F., KURVARI, V., VIRMANI, A., GOLLAHON, L., SAKAGUCHI, M., WESTERFIELD, M., KODAGODA, D., STASNY, V., CUNNINGHAM, H. T., WISTUBA, I. I., ET AL. Characterization of paired tumor and non-tumor cell lines established from patients with breast cancer. *International journal of cancer 78* (1998), 766–774.

[8] GROSSMAN, M., BRETERNITZ, M., AND SARKAR, V. HadoopCL2: Motivating the design of a distributed, heterogeneous programming system with machine-learning applications. *Parallel and Distributed Systems, IEEE Transactions on PP*, 99 (2015), 1–1.

[9] HUANG, M., AND WU, D. Deploying accelerators at datacenter scale using Spark. *Spark Summit 2016* (2016).

[10] INTEL. Intel-Altera heterogeneous architecture research platform (HARP) program. URL: http://www.sigarch.org/2015/01/17/call-for-proposals-intel-altera-heterogeneous-architecture-research-platform-program/.

[11] JASON CONG, MUHUAN HUANG, D. W. C. H. Y. Heterogeneous datacenters: Options and opportunities. *Design Automation Conference* (2016).

[12] LANGMEAD, B., AND SALZBERG, S. L. Fast gapped-read alignment with Bowtie 2. *Nature methods 9*, 4 (2012), 357–359.

[13] LI, H., AND DURBIN, R. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics 25*, 14 (2009), 1754–1760.

[14] LI, P., AND LUO, Y. HeteroSpark: A Heterogeneous CPU/GPU Spark Platform for Deep Learning Algorithms. *Spark Summit 2015e* (2015).

[15] MCKENNA, A., HANNA, M., BANKS, E., SIVACHENKO, A., CIBULSKIS, K., KERNYTSKY, A., GARIMELLA, K., ALTSHULER, D., GABRIEL, S., DALY, M., ET AL. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research 20*, 9 (2010), 1297–1303.

[16] NOTHAFT, F. A., MASSIE, M., DANFORD, T., ZHANG, Z., LASERSON, U., YEKSIGIAN, C., KOTTALAM, J., AHUJA, A., HAMMERBACHER, J., LINDERMAN, M., FRANKLIN, M. J., JOSEPH, A. D., AND PATTERSON, D. A. Rethinking data-intensive science using scalable analytics systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 631–646.

[17] PUTNAM, A., CAULFIELD, A. M., CHUNG, E. S., CHIOU, D., CONSTANTINIDES, K., DEMME, J., ESMAEILZADEH, H., FOWERS, J., GOPAL, G. P., GRAY, J., ET AL. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (2014), ISCA '14, pp. 13–24.

[18] SHENDURE, J., AND JI, H. Next-generation DNA sequencing. *Nature biotechnology 26*, 10 (2008), 1135–1145.

[19] SMITH, T., AND WATERMAN, M. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 1 (1981), 195 – 197.

[20] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (2012), NSDI'12, pp. 2–2.

[21] ZHANG, P., TAN, G., AND GAO, G. R. Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform. In *Proceedings of the 1st International Workshop on High-performance Reconfigurable Computing Technology and Applications* (2007), HPRCTA '07, ACM, pp. 39–48.