Towards Exascale Computing with Heterogeneous Architectures

Kenneth O'Brien[1], Lorenzo Di Tucci[2], Gianluca Durelli[1], Michaela Blott[1]
[1]Xilinx Research Labs, Dublin, Ireland,
[2]Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Milano, Italy,
lorenzo.ditucci@mail.polimi.it
{*kenneth.o'brien,mblott,gdurelli*}@xilinx.com

***Abstract*—The goal of reaching exascale computing is made especially challenging by the highly heterogeneous nature of modern platforms and the energy they consume. As compute nodes typically utilize multiple multi-core CPU and are increasingly equipped with PCIe based accelerators, both are contributing to an ever more dynamic power consumption. In our study we evaluate our target application on a variety of heterogeneous platforms, including high end FPGA, GPU, and Xeon Phi accelerators, with respect to energy efficiency at a node and cluster level. We compare multiple implementations of our application, each built with a different modern parallel programming framework, with respect to execution performance, code complexity and energy efficiency. Later we extrapolate based on our findings, the implications of scaling this application towards exascale, with projections of computation achievable within the exascale power budget for our three architectures.**

I. Introduction

Exascale computing is critical to solve many of the world's currently unsolvable problems ranging from genomics, where we have the potential to save lives through active cancer research, to climate modeling, where potentially large-ensemble multi-decadel predictions could help us to prepare for the future. These applications drive enormous compute burdens from hundreds of peta flops to hundreds of exa flops [1], while storage requirements escalate. For example, according to [2], genomics workflows will require an estimated 10 petabyte in FY2021, and climate modeling, one hundred exabyte. Furthermore, operating costs for HPC centers as well as their carbon footprint need to be addressed. As a result power, energy-efficiency, and cooling become first-class constraints for scalable HPC.

On the other side, our conventional von-Neumann architectures are suffering from limited performance scalability and increasing power densities as described in detail in [3]. A new era in computing emerges which spawns architectural innovation in form of the adoption of increasingly heterogeneous computing architectures embracing accelerators such as Graphics Processing Unitss (GPUs) and Field Programmable Gate Arrays (FPGAs) to provide the much sought after performance scalability and power reduction. GPGPUs are attractive for their high floating point performance that can be achieved by a highly parallel SIMD architecture. FPGAs can bring benefit as they can tailor the hardware to the application through customized datapaths, operators, data types and memory architectures. Thereby FPGAs can achieve much higher energy efficiencies compared to conventional CPU- and GPU-based solutions. This has stimulated interest in their exploitation within data centers [4] with recent benchmarks showing that FPGA-based application acceleration can bring orders of magnitude improvement in regards to performance and performance per Watt compared to CPU/GPU counterparts [5]–[8] for a broad range of applications, from machine learning to graph traversal to genome sequencing.

However, typically this work has been implemented through cumbersome, hardware-centric RTL design flows, the traditional FPGA development path, which is not really accessible to larger parts of the HPC community which are used to a different more abstract design environment and requires very different skill sets. Driven by this need, new software-centric design environments are emerging that can tremendously boost the productivity of designers and open up FPGA acceleration to masses of software engineers [9]. The opportunity lies in automating the architectural optimizations that are required to achieve performance scaling with custom hardware architectures implemented in FPGAs beyond standard pipelining and loop unrolling techniques.

In this paper, we take an in-depth look at a specific algorithm which is highly relevant within the context of cancer research. We carry out elaborate test benching on latest Nvidia GPUs, Intel Xeon Phi devices as well state-of-the-art FPGAs for a class of different inputs to see what what compute architecture delivers best characteristics across the given set of figure of merits, including absolute performance, power consumption and code complexity. For single-node deployments, GPU outperform all other devices when multiple independent queries are used while FPGAs are better for power efficiency which is the target for exascale research. Contrary to popular belief, FPGA code is significantly simpler in comparison to GPUs utilizing latest design entry techniques.

II. Motivation and Case Study

In this work, we use as a case study an application from the biomedical domain which is of utmost importance in the context of genome and cancer research. In particular,

978-3-9815370-8-6/17/$31.00

we consider the problem of finding the sequence alignment between two strings that can be nucleotides. Techniques for efficiently solving this problem have proven to be important tools to biomedical researchers, however the availability of an increasing volume of genomic data is putting a lot of pressure on the High Performance Computing (HPC) community to further improve the performance.

In particular this work considers the Smith-Waterman [10] algorithm used to solve the sequence alignment problem. While this algorithm is designed for cancer research, as similarity search algorithm it has a much wider applicability. For instance, it can be exploited to perform fingerprint analysis or to identify web-page duplicates [11].

As a fundamental application in the field, there have been many attempts to improve its performance. In particular multiple approaches, exploiting different accelerators (Central Processing Unit (CPU), GPU, FPGA), have been proposed over the last few years [12]–[15]. This heterogeneity, combined with the continuing increase in the amount of data used as a reference database, makes this application a perfect fit for the analysis performed in this paper. It is in fact easy to foresee that, in the next few years, common solutions based on multicores CPU and even GPU will not be able to provide an acceptable level of performance while meeting a reasonable power budget.

This combined effort in optimizing performance and power budget is in fact what characterizes the Exascale research. In this context custom hardware accelerators will play a key role as we will show in the remainder of our analysis. In this work we analyze three different implementations available in the literature of the Smith-Waterman algorithm. The first one is a GPU implementation of the algorithm [14] which makes use of the high thread and data level parallelism available on GPUs to reach very high performance, while at the same time having the drawback of an high power consumption. The second solution targets a Xeon Phi coprocessor [13] that exploits a many core architecture. While the last solution we analyzed uses an FPGA exploiting instruction level parallelism [16]. These three solutions are compared using different metrics (i.e. pure performance, energy efficiency, and code complexity) on a single node machine and then we extrapolate insights on cluster scaling and what will then be happening in an exascale context.

A. Algorithm description

The Smith-Waterman algorithm is a dynamic programming algorithm that performs pairwise local sequence alignment between two strings, called the *query* and the *database*. The main difference among the Smith-Waterman algorithm and other algorithms is that it does not exploit heuristic mechanisms; consequently the alignment produced is guaranteed to be the optimal one with respect to the considered scoring system [17]. This algorithm takes as input two strings, the query and the database, and identifies how the query sequence aligns to the database. The processing steps to obtain the alignment are mainly two. The first one involves the computation of two matrices, namely the *similarity* (*S*) and the *traceback* (*T*) matrix, and the identification of the maximum element in the similarity matrix. The second one performs the real alignment, following the directions stored in the traceback matrix.

1) Matrices Computation: Given two strings, namely the query and the database sequence, and a scoring system, this first step of the algorithm computes a similarity and traceback matrix. The first one holds the *scores* of the comparison of each element of the string with the database sequence. The scores are calculated using the provided *scoring system*. Each element of the similarity matrix is calculated as in Equation (1).

$$S(i,j) = max \begin{Bmatrix} S(i-1,j-1) + s(\mathcal{N}_i, \mathcal{M}_j) & Match/Mismatch \\ max_{k \geq 1} S(i-k,j) + gap_{del} & Deletion \\ max_{k \geq 1} S(i,j-l) + gap_{ins} & Insertion \\ 0 \end{Bmatrix} \quad (1)$$

where

$$1 \leq i \leq n, 1 \leq j \leq m \quad (2)$$

In Equation (1) gap_{del} and gap_{ins} represent the gap-scoring scheme in case of deletion or insertion, $s(\mathcal{N}_i, \mathcal{M}_i)$ is a similarity function over the two considered elements of the two strings, $\mathcal{N}$ represents the query, $\mathcal{M}$ represents the database, n is the size of the query string and m is the size of the database. For each element stored into the similarity matrix, we store also a value in the traceback matrix. The traceback matrix is created to hold the directions that will be used to create a valid alignment. Each element in the traceback matrix identifies the direction of the element representing the maximum score in Equation (1). As the maximum is calculated over 4 values, there are 4 possible directions, namely *north*, *north-west*, *west* and *center*. During the computation of this two matrices, the algorithm keeps track of the maximum element in the similarity matrix, as well as its index. This values will be used for the following step of the algorithm.

2) Traceback: Once the two matrices have been computed and the index of the maximum element identified, the algorithm can perform the final alignment. In this step, the algorithm starts from the element in the traceback matrix identified by the index of the maximum element of the scoring matrix and follows the directions stored to find the optimum alignment. The algorithm keeps following the directions stored until a *center* is found. Then it returns how the query string aligns to the database.

From a high level analysis of the algorithm it is understandable that it is a good candidate for an implementation on parallel machines. The first step of the algorithm is highly parallel as each element that is located on an anti-diagonal of the similarity matrix is dependency-free, once the elements of the two previous anti-diagonals have been computed. This means that it is possible to compute independently and at the same time the elements of each anti-diagonal. In the case of the second step of the algorithm, the level of parallelism exploitable is very low. As each direction stored in the traceback matrix provides information of the next direction to follow, this step must be executed sequentially.

III. Single Node Analysis

In this section we discuss our experimental apparatus, the results of our performance, power and energy evaluation, complexity of the code for each implementation and finally our conclusions of the single node analysis.

A. Experimental Platform

In all experiments, the Smith-Waterman algorithm was accelerated by a PCIE based accelerator card, all of which are high performance computing grade. Below we describe the specifications of each accelerator.

The FPGA accelerator board used in our experiments was the ADM-PCIE-KU3 by AlphaData. It is equipped with a Kintex Ultrascale KU060 FPGA device with 331,680 LUTS, 2,760 DSP slices, 663,360 flip-flops, and 38MB of onchip memory in the form of so called block RAM (BRAM). The board also hosts 2x8GB DDR3 ECC memory with a theoretical peak memory bandwidth of 25.6GB/s. The peak power consumption for the board is 25W. For all experiments, our designs were executed with a clock frequency of 250Mhz. The Smith-Waterman implementation used was developed by the authors using Xilinx's OpenCL development environment SDAccel and is fully described in [16].

For our Xeon Phi experiments, we used a Xeon Phi 3120P, with 57 cores running at 1.1Ghz. This device is equipped with 6GB of GDDR5 ECC memory with a peak theoretical memory bandwidth of 240GB/s. The device has a rated peak power consumption of 300W. The software implementation instrumented on this device was SWAPHI-LS [13] and is implemented using a combination of OpenMP for thread parallelism and AVX instrinsics for SIMD vector parallelism.

Finally, the GPU in our experiments was the Nvidia Tesla K40c. This Kepler architecture device hosts 15 SMX, running at 745Mhz. The board hosts 12GB of GDDR5 ECC memory with a peak theoretical memory bandwidth of 288GB/s. The peak power consumption of this device is 235W. The software implementation in the case of the GPU was SW# [15], [18], which is implemented with Nvidia CUDA.

B. Performance and Energy Efficiency

In our experiments, we compared the performance of all three accelerator devices executing the Smith-Waterman algorithm on inputs of varying $\mathcal{M}$ and $\mathcal{N}$. Each device was sent the same synthetic FASTA format input files, a query of length n and a single database entry of length m. The results are shown in table I.

We report measurements derived only from the execution time on the device, excluding memory allocation and data transfer with the host. In the case of the FPGA, OpenCL events are used to measure this value. For the Xeon Phi, we report the value produced by the SWAPHI-LS tool, and for the GPU, we report the kernel execution time as reported by the Nvidia CUDA profiler nvprof. All performance measurements are reported in units of Giga Cell Updates per Second (GCUPS), which is the number of cells of the similarity matrix (in billions) being updated per second as done so in the literature.

TABLE I
GCUPS Performance of GPU, Xeon Phi and FPGA with varying input sizes

n	m	**GPU**	**Xeon Phi**	**FPGA**
128	16384	0.3013	0.253278	5.737
128	131072	0.2873	2.446417	6.26044
128	1048576	0.1809	11.180068	6.3516
256	16384	2.4154	0.590401	30.487
256	131072	0.5754	5.136217	40.069
256	1048576	0.35732	22.183034	42.4216

The formula for converting execution time to GCUPS is presented in equation 3, where t is execution time in seconds.

We measured performance of one query against one database entry of varying sizes. At this task the FPGA achieves the best performance with a 1.9x-118x speedup over the other devices (as shown in table I). The performance advantage here is due to the FPGA's high degree of pipeline parallelism, allowing the FPGA to constantly read, compute and write the results in parallel.

In the case of the GPU implementation the performance decreases in both cases of an increasing m and n due to the worsening ratio of m to n, leading to an inability to exploit the resources of the device. This behavior is known to the developers of SW#. For the Xeon Phi, the performance improves due to the increasing partitioning of data in parallel across threads and SIMD units.

Due to the data parallel nature of the GPU and Xeon Phi, they derive high performance from scheduling many query/database lookups in parallel exploiting thread level parallelism. This configuration is typical of a production deployment when multiple query/databases have to be performed at the same time. To fully utilise the FPGA implementation in this context, we would queue multiple lookups sequentially to the FPGA device.

To better compare our FPGA solution with the other two architectures we rely on published best results. For the GPU we used the data in [14], while for Phi we rely on [12], [13]. Both of these works demonstrate the performance achievable with the Smith-Waterman algorithm in a typical production environment. In figure 1, we show the peak measured performance for the three devices. As we can see, if we consider a real use case scenario instead of the single query/database alignment, the FPGA is now having the lowest performance (42.42 GCUPS) compared to the Phi (58.8 GCUPS) and the GPU (169.7 GCUPS).

To derive worst case energy efficiency, we used the thermal design power (TDP) for the three platforms which are 25W for the FPGA, and 300W for both Phi and GPU. fig. 2 shows the results of the energy efficiency comparison. With the TDP of an FPGA being 6 times lower than any of the other devices, the resulting energy efficiency of the FPGA solution outperforms GPU and Phi counterparts by more than a factor of 3x.

$$GCUPS = \frac{\frac{m*n}{t}}{10^9} \tag{3}$$

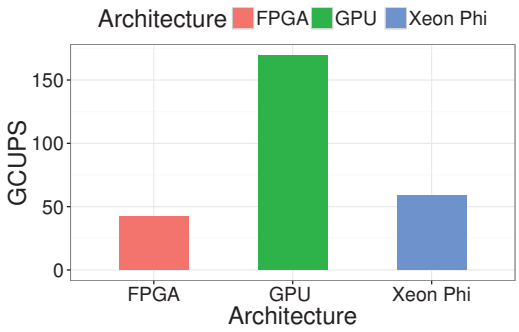

Fig. 1. Best Reported Performance for each Architecture

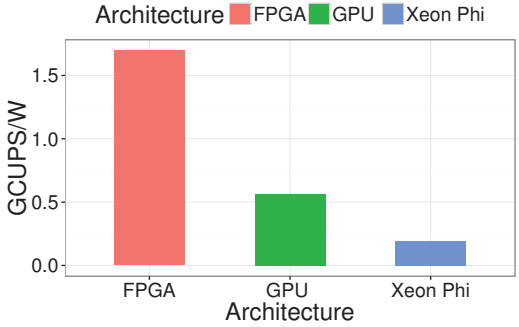

Fig. 2. Energy Efficiency Based on Best Reported Performance for each Architecture

C. Code Complexity

In this section we discuss the various implementation design entries.

In the case of the FPGA implementation, the kernel is implemented using the Vivado High Level Synthesis C/C++ design entry. This level of abstraction allows the developer to express efficient hardware architectures using comparatively less code than traditional RTL based designs, on a par with existing accelerator devices as seen in table II. The final implementation, was completed using the SDAccel framework, that abstracts the system design step, increasing the productivity while leveraging high performance.

The GPU implementation, is developed with Nvidia CUDA. The CUDA framework is nowadays the most widely used one thanks to the fact that it is generally provides a greater suite of vendor optimized libraries when compared to other GPGPU programming paradigms such as OpenCL.

SWAPHI-LS [13], the implementation used on the Xeon Phi, was developed with a combination of AVX intrinsics and OpenMP pragmas. The former facilitates access to the SIMD vector instructions, while the latter brings thread parallelism across the 57 cores.

By comparing number of lines of code used, reported in table II, we can see how, contrary to common belief, the FPGA is not the solution requiring the most code. The reason behind this comes from the availability of new tools that automatically perform translation from C/C++ language to the RTL description. The shortest code is the Phi's one since OpenMP and AVX intrinsics allow to perform what are usually complex operations in few lines of code. Finally, the GPU requires $\tilde{2}$x lines of code compared to the other two since to extract good performance on a GPU you need to adapt the code to GPU architecture and memory hierarchy.

TABLE II
LINES OF CODE PER IMPLEMENTATION

Device	Lines of code
FPGA	194
GPU	371
Xeon Phi	177

D. Discussion

As shown in [19], a specific device can be shown to give the best performance for a particular application. For this application, the device yielding the highest performance depends on m, n, the volume of and ratio of queries to databases. GPU and Phi need multiple query/database alignment in order to express their full potential. Looking at the results presented we see an increase of performance of 70x when moving from a single query to a fully loaded GPU (2.5x in the case of the Xeon Phi). On the contrary our FPGA solution has good performance for a single alignment, when compared to the other two architectures, and keeps that same performance when multiple alignments are required due since they will be executed in a queue.

In general the FPGA solution guarantees a more consistent performance which does not depend on the number of alignments performed, but has the lowest top performance when the system is kept fully loaded. However, despite the lowest performance, the FPGA is able to obtain the best power efficiency compared to the other solutions.

IV. From Clusters to Exascale

In this section, we describe the process of moving the various implementations to a multiple node cluster.

The first issue in scaling is to make effective use of multiple acceleration devices in the same node. In the Smith-Waterman application, query to database lookups are independent. Assuming each compute device has a full copy of the database, we can partition the queries between the devices. In the case of all three devices, the offload is managed by the host CPU runtime. For GPU this is the CUDA runtime, for Xeon Phi it is the MIC runtime and for the FPGA, we have used OpenCL. Given the current longest sequenced DNA transcript has a length of 20 billion base pairs [20] and current efforts target a 35 billion base pair organism, storing the whole database on each device is in that case practical only on for the FPGA implementation, due to the high memory density of off-chip memory and the efficient 2bit compressed representation of the data [16]. As the data cannot be efficiently processed in this format on the other two devices, the FPGA brings a scaling benefit to this application.

Current approaches to large scale distributed computing are centered on MPI. MPI provides a C API for message passing between distributed processes within an application. These messages carry data between processes or control synchronization. The API supports point-to-point and collective operations which facilitate communication of data from all to all (MPI_Alltoall), from all to one (MPI_Gather) and from one to all (MPI_Scatter) processes. Though MPI can pass messages between processes operating on the same host through shared memory, applications developed with MPI frequently use another parallel programming framework such as OpenMP to parallelize at the thread level. MPI can also be used in conjunction with accelerator devices through the use of OpenCL, CUDA and Xeon Phi offload.

Both swsharp and swaphi-ls have an MPI capability to partition a workload across multiple nodes. In [15] and [13] we see how the problem scales near linearly to multiple devices using the MPI capability. From here we project the best performance of the applications for each device from figure 1 using the scaling seen in [13], [15]. Up until now, we have considered each accelerator as a standalone device from an energy efficiency perspective. In practice we must consider the effect of the system hosting these devices on the overall energy efficiency of the nodes and total system at large.

In figure 3 we extrapolate for each device, the energy efficiency of a node using a varying number of each accelerator for three different host systems of varying base power consumption: 100W, 200W, and 300W. We consider this base power to represent only the wall power and we do not consider the power consumed by cooling and interconnect infrastructure. As shown in figure 3, the energy efficiency of a node with $2^0 \ldots 2^4$ Xeon Phi is saturated once four accelerators are included. For the GPU, a better energy efficiency is achieved of 0.5 GCUPS/W, but the value is saturated also after four compute accelerators. For the case of the FPGA, a high energy efficiency of 1.36 is achieved with 16 accelerators and does not saturate until 256 FPGA are in the single node. Consolidating more accelerators into a single node is also beneficial in terms data locality inside the datacenter.

The single node analysis assumed that the multiple accelerators per node are connected by means of PCIE connection, while at datacenter scale we need to take into account multiple nodes communicating via the network infrastructure. In the remainder of the discussion we assume linear scaling when moving from one node to multiple nodes. Although this assumption might be seen as too simple, we have to keep in mind that for Smith-Waterman algorithm we only need to communicate to the multiple nodes the query, which is generally a short string and after that the compute intensive part can execute without any synchronization. Communicating a set of short strings is unlikely to saturate the network bandwidth available in today datacenters, so linear scaling in this case may be a reasonable assumption to perform our analysis.

Our first analysis consists in identifying, starting from the possible nodes configurations identified, what is the performance currently attainable considering the power cap of 20MW identified for Exascale computation. Figure 4 illustrates the performance attainable by replicating a node configuration as many time as possible while respecting the power cap. Three plots reports the solutions for the three different node base power. As we can see the low power efficiency of the Xeon Phi lead to a poor result at scale for this architecture, less than 5 PCUPS. The high power consumption of the GPU does not allow to instantiate a high number of nodes, but the higher performance of this architecture allow to possibly attain more than 10 PCUPS with today's hardware. Finally we can see that the high power efficiency of the FPGA, coupled with the lower power footprint of the nodes (when consolidating more than 4 or 8 FPGAs), would allow us to obtain more than 25 PCUPS in the best scenario (base node power of 100W) and 20 PCUPS in the worst one. From this analysis it is clear how, in this specific application, FPGA is a promising architecture to target the exascale scenario.

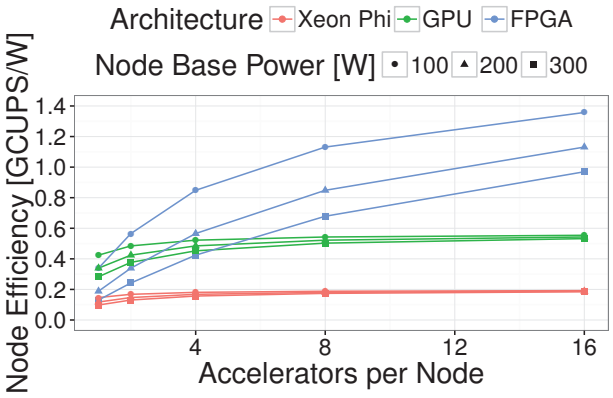

Fig. 3. Node Power Efficiency with varying numbers of accelerators

TABLE III
PERFORMANCE, POWER CONSUMPTION AND PERFORMANCE/W/NODE FOR EACH DEVICE

Architecture	GCUPS/Node	W/Node	GCUPS/W/Node
FPGA	678.75	500	1.36
GPU	2715.20	4900	0.55
Xeon Phi	940.80	4900	0.19

Finally we show the minimum number of nodes required to attain one Exa CUPS, and the total power they would consume in table IV. As the table shows the FPGA is the architecture with the lowest power footprint at scale, but the actual footprint is significantly higher than the 20MW power cap that exascale research wants to obtain. From this we can conclude that, even if the FPGA is currently the best alternative for this particular solution, the research to achieve exascale is still very open.

V. CONCLUSION

In this paper, we have explored the performance, energy efficiency and code complexity of the Smith-Waterman algorithm using the best known measurements. Overall, we found the GPU to have the best performance, but the FPGA

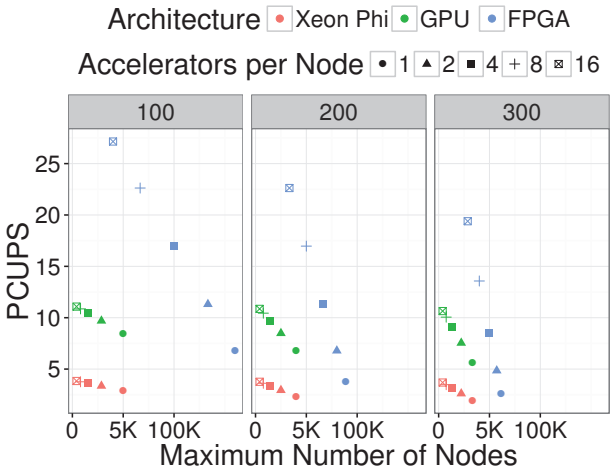

Fig. 4. Predicted Performance Achievable within 20MW budget for configurations found in Figure 3

TABLE IV
#NODES REQUIRED FOR EXACUP COMPUTATION AND TOTAL POWER ESTIMATE

Architecture	#Nodes for ECUP	Total Power(MW)
FPGA	1473306	736.65
GPU	368297	1804.66
Xeon Phi	106295	5208.33

to demonstrate the best energy efficiency within a single node deployment. When scaling this algorithm over multiple accelerator devices to a multinode cluster, we are taking into account the base power of the host system. For this scenario, we found that the FPGA implementation provided the best scalability to high numbers of accelerators. We extrapolated to find that for host systems with three example base power consumption, what implementation would provide the best performance within the 20MW exascale power envelope. Furthermore, the FPGA solution offers the highest performance followed by the GPU and PHI, although the FPGA required overall the highest numbers of individual nodes to achieve the performance. Lastly, given the translation of a CUP operation on the FPGA to be 32 operations, the FPGA solution would achieve an exa operation of this application using 46040 nodes in 736MW. In future, we will develop our implementation to support multiple pipelined alignments, extend our design to support protein alignment and work towards a distributed multi-FPGA implementation.

REFERENCES

[1] A. Sodani and C. Processor, "Race to exascale: Opportunities and challenges," in *Keynote at the Annual IEEE/ACM 44th Annual International Symposium on Microarchitecture*, 2011.

[2] Kusnezov, Binkley, Harrod, and Meisner. (2013, Sep.) Doe exascale initiative. [Online]. Available: http://energy.gov/sites/prod/files/2013/09/f2/20130913-SEAB-DOE-Exascale-Initiative.pdf

[3] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.

[4] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 13–24.

[5] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 47–56.

[6] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martínez, and C. Guestrin, "Graphgen: An fpga framework for vertex-centric graph computation," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*. IEEE, 2014, pp. 25–28.

[7] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers, and R. Carley, "Scalable 10gbps tcp/ip stack architecture for reconfigurable hardware," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 36–43.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[9] G. Guidi, E. Reggiani, L. D. Tucci, G. Durelli, M. Blott, and M. D. Santambrogio, "On how to improve fpga-based systems design productivity via sdaccel," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 247–252.

[10] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195 – 197, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022283681900875

[11] S. Xu, D. A. Smith, A. Mullen, and R. Cordell, "Detecting and evaluating local text reuse in social networks," *ACL 2014*, p. 50, 2014.

[12] Y. Liu and B. Schmidt, "Swaphi: Smith-waterman protein database search on xeon phi coprocessors," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, June 2014, pp. 184–185.

[13] Y. Liu, T. T. Tran, F. Lauenroth, and B. Schmidt, "Swaphi-ls: Smith-waterman algorithm on xeon phi coprocessors for long dna sequences," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2014, pp. 257–265.

[14] Y. Liu, A. Wirawan, and B. Schmidt, "Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions," *BMC Bioinformatics*, vol. 14, no. 1, p. 117, 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-14-117

[15] M. Korpar and M. iki, "Sw#gpu-enabled exact alignments on genome scale," *Bioinformatics*, vol. 29, no. 19, pp. 2494–2495, 2013. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/29/19/2494.abstract

[16] L. Di Tucci, K. O'Brien, M. Blott, and M. D. Santambrogio, "Architectural Optimizations for High Performance and Energy Efficient Smith-Waterman Implementation on FPGAs using OpenCL," in *2017 Design, Automation and Test in Europe*. IEEE, Accepted to appear, 2016.

[17] E. Roberts. (2016, Nov.) Smith-waterman algorithm. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/computers-and-the-hgp/smith_waterman.html

[18] M. Korpar, M. oi, D. Blaeka, and M. iki, "Sw#db: Gpu-accelerated exact sequence similarity database search," *PLOS ONE*, vol. 10, no. 12, pp. 1–11, 12 2016. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0145857

[19] S. Muralidharan, K. O'Brien, and C. Lalanne, "A semi-automated tool flow for roofline anaylsis of opencl kernels on accelerators," *First International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC '15)*, 2015.

[20] J. L. Wegrzyn, J. D. Liechty, K. A. Stevens, L.-S. Wu, C. A. Loopstra, H. A. Vasquez-Gross, W. M. Dougherty, B. Y. Lin, J. J. Zieve, P. J. Martínez-García, C. Holt, M. Yandell, A. V. Zimin, J. A. Yorke, M. W. Crepeau, D. Puiu, S. L. Salzberg, P. J. de Jong, K. Mockaitis, D. Main, C. H. Langley, and D. B. Neale, "Unique features of the loblolly pine (pinus taeda l.) megagenome revealed through sequence annotation," *Genetics*, vol. 196, no. 3, pp. 891–909, 2014. [Online]. Available: http://www.genetics.org/content/196/3/891