An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm

Houtgast, Ernst; Sima, VM; Bertels, K; Al-Ars, Z

DOI
10.1109/SAMOS.2015.7363679

Publication date
2015

Document Version
Accepted author manuscript

Published in
Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XV

Citation (APA)
Houtgast, E., Sima, VM., Bertels, K., & Al-Ars, Z. (2015). An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. In D. Soudris, & L. Carro (Eds.), *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XV* (pp. 221-227). IEEE. https://doi.org/10.1109/SAMOS.2015.7363679

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

An FPGA-Based Systolic Array to Accelerate the BWA-MEM Genomic Mapping Algorithm

Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels and Zaid Al-Ars
Faculty of EEMCS, Delft University of Technology, Delft, The Netherlands
E-mail: {e.j.houtgast, v.m.sima, k.l.m.bertels, z.al-ars}@tudelft.nl

***Abstract*—We present the first accelerated implementation of BWA-MEM, a popular genome sequence alignment algorithm widely used in next generation sequencing genomics pipelines. The Smith-Waterman-like sequence alignment kernel requires a significant portion of overall execution time. We propose and evaluate a number of FPGA-based systolic array architectures, presenting optimizations generally applicable to variable length Smith-Waterman execution. Our kernel implementation is up to 3x faster, compared to software-only execution. This translates into an overall application speedup of up to 45%, which is 96% of the theoretically maximum achievable speedup when accelerating only this kernel.**

I. Introduction

As next generation sequencing techniques improve, the resulting genomic data, which can be in the order of tens of gigabytes, requires increasingly long time to process. This is becoming a large bottleneck, for example in cancer diagnostics. Hence, acceleration of algorithms used in genomics pipelines is of prime importance. General purpose processors are not necessarily the best execution platform for such workloads, as many bioinformatics workloads lend themselves well to parallel execution. Use of dedicated hardware, such as GPUs or FPGAs, can then greatly accelerate the computationally intensive kernels to achieve large speedups.

The initial stage for many genomics pipelines is sequence alignment. DNA sequence reads are aligned against a reference genome, producing the best found alignment for each read. Many sequence alignment tools exist, such as Bowtie [5], BWA [8], MAQ [9], and SOAP2 [10]. BWA-MEM [7] is widely used in practice as a de facto sequence alignment algorithm of choice. In this paper, we investigate and propose the first accelerated version of BWA-MEM, using FPGAs to improve performance. The use of FPGAs can yield order-of-magnitude improvements in both processing speed and power consumption, as they can be programmed to include a huge number of execution units that are custom-tailored to the problem at hand, providing much higher throughput than conventional processors. At the same time, they consume much less power, since they operate at relatively low clock frequencies and use less silicon area for the same task.

In this paper, we present the following contributions: we (1) analyze the BWA-MEM algorithm's main execution kernels; (2) propose novel systolic array design approaches optimized for variable length Smith-Waterman execution; (3) implement and integrate the design into the BWA-MEM algorithm; and thus (4) create the first accelerated version of the BWA-MEM algorithm, using FPGAs to offload execution of one kernel.

The rest of this paper is organized as follows. Section II provides a brief background on the BWA-MEM algorithm. Section III describes the details of our acceleration approach. Section IV discusses design alternatives for the systolic array implementation. Section V provides the details on the configuration used to obtain results, which are presented in Section VI and then discussed in Section VII. Section VIII concludes the paper and indicates directions for future work.

II. BWA-MEM Algorithm

The BWA program is *"a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM ... BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate."* [6] A characteristic workload of this algorithm is to align millions of DNA reads against a reference genome. Currently, we align DNA reads of 150 base pairs (bp), a typical output length of next generation sequencers [1], against the human genome.

A. BWA-MEM Algorithm Kernels

The BWA-MEM algorithm alignment procedure consists of three main kernels, which are executed in succession for each read in the input data set.

1. SMEM Generation: Find likely mapping locations, which are called *seeds*, on the reference genome. To this end, a BWT-based index of the reference genome, which has been generated beforehand, is used [8]. Per read, zero or more seeds are generated of varying length;

2. Seed Extension: Chain and extend seeds together using a dynamic programming method that is similar, but not identical, to the Smith-Waterman algorithm [12];

3. Output Generation: Sort and, if necessary, perform global alignment on the intermediate results and produce output in the standardized SAM-format.

The BWA-MEM algorithm processes reads in *batches*. Figure 1 illustrates the order in which these kernels process a batch of reads. The first two kernels, SMEM Generation and Seed Extension, are performed directly after each other for each read. When this is finished for all reads in a batch, Output Generation is performed. BWA-MEM implements multi-threaded execution of all three program kernels.

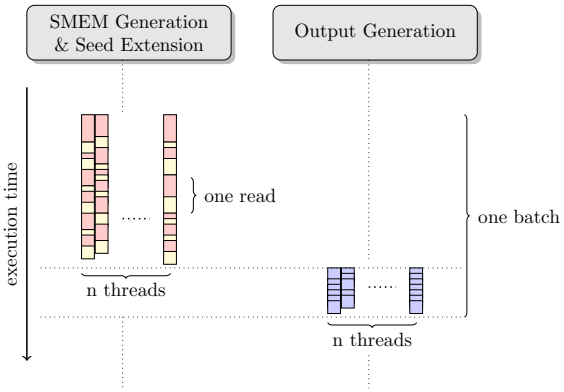

Fig. 1. Execution order of the three main BWA-MEM algorithm kernels. Per batch, execution of SMEM Generation and Seed Extension is intertwined for each read; afterwards, Output Generation is performed.

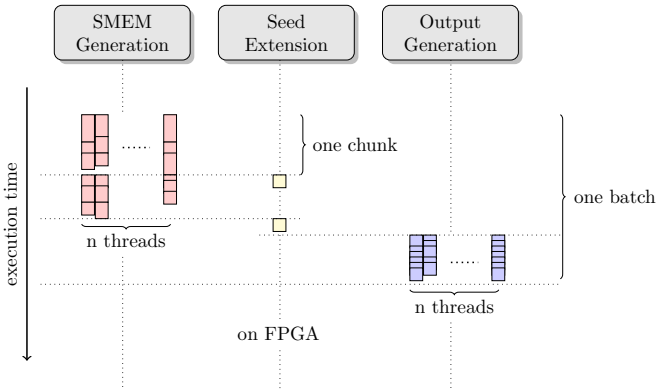

Fig. 2. Our implementation further subdivides batches into chunks. SMEM Generation and Seed Extension are separated and Seed Extension is mapped onto the FPGA; its execution is overlapped with SMEM Generation.

B. Profiling Results

A challenging factor in the acceleration of the BWA-MEM algorithm is the fact that execution time is not isolated in a single computationally-intensive kernel, but is spread over three separate kernels. Hence, speedup of a single kernel will only yield limited overall speedup. To investigate which of these kernels is most suitable for FPGA-based acceleration, we have analyzed and profiled the algorithm with `gprof` and `oprof`. Both yielded similar results. Table I shows the profiling results for a typical workload.[1]

TABLE I
RESULTS OF BWA-MEM ALGORITHM PROFILING

Program Kernel	Time	Bottleneck	Processing
SMEM Generation	56%	Memory	Parallel
Seed Extension	32%	Computation	Parallel
Output Generation	9%	Memory	Parallel
Other	3%	I/O	Sequential

For each kernel, the relative execution time, type of processing and whether it is bound by computation, memory or I/O is specified, based on a combination of profiling and source code inspection. Besides these computationally-intensive kernels, the remaining execution time is comprised of other activities, among them initialization and I/O.

In this paper, we investigate acceleration of the Seed Extension kernel. This work is part of an on-going effort to accelerate execution of the BWA-MEM algorithm. Our rationale to start with the Seed Extension kernel is as follows: although profiling results indicate that the SMEM Generation kernel is more time-consuming, the dynamic programming-type of algorithm used in the Seed Extension kernel is a much better fit for execution on an FPGA. As shown in Table I, Seed Extension requires 32% of overall execution time for this workload. Hence, the maximum speedup we can expect to gain from accelerating this part is 47%.

[1]Single-ended GCAT on the Convey HC-2^{EX} (refer to Section V for more details on the workload and execution platform).

III. IMPLEMENTATION

Our efforts to accelerate the Seed Extension kernel can be divided into two parts: (1) the FPGA-accelerated core that implements the Seed Extension kernel; and (2), integration of this kernel into the BWA-MEM algorithm.

A. Top-Level Accelerated Structure

As shown in Section II-B, the BWA-MEM algorithm consists of three distinct kernels. As illustrated in Figure 1, execution of the SMEM Generation kernel and Seed Extension is intertwined. Directly using this structure to offload execution of the Seed Extension kernel onto an FPGA would require a large number of small data transfers, two per read. The resulting overhead makes such a structure undesirable.[2] Hence, in order to facilitate offloading this kernel onto the FPGA, SMEM Generation and Seed Extension are completely separated from each other, which allows for fewer, but larger transfers of data to and from the FPGA. The modified structure of operation is shown in Figure 2. This approach does require that some temporary data structures are kept in memory longer. In practice, this is not an issue as the data is in the order of tens of megabytes.

The accelerated approach is based on two principles: (1) offloading the Seed Extension kernel onto the FPGA; and (2) overlapping execution of SMEM Generation on the host CPU and Seed Extension on the FPGA, thereby effectively hiding the time required to process this stage. In order to overlap these kernels, the reads in a batch are further subdivided into *chunks*. After a chunk is processed by the SMEM Generation kernel, it is dispatched to the FPGA. Output Generation is performed only after both the kernels finish, as the CPU cores are fully utilized while performing SMEM Generation. Hence, there would be no benefit in overlapping execution of this kernel with the other two.

Reads vary in the amount of temporary data required to process them: some reads generate more SMEMs (i.e., potential

[2]For example, testing reveals that copying 1 Mbyte at once is almost 30x faster than performing a thousand 1 kbyte transfers.

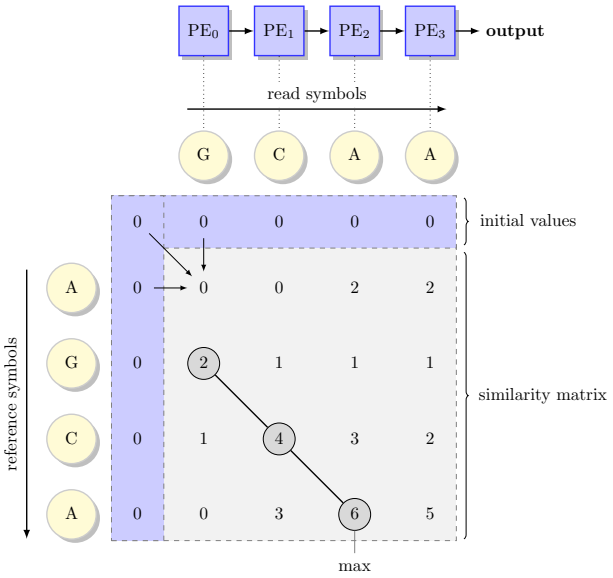

Fig. 3. Smith-Waterman similarity matrix showing the local sequence alignment with maximum score. Each read symbol is mapped onto a Processing Element of the systolic array.

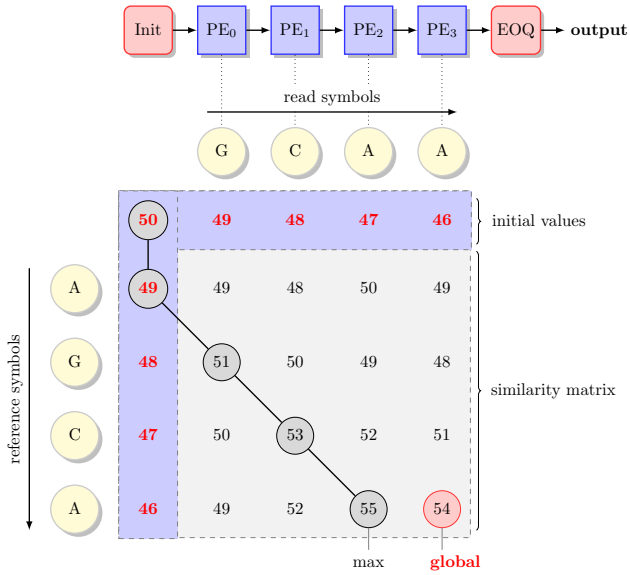

Fig. 4. Seed Extension similarity matrix showing an extension with an initial score of 50. The implications to the systolic array design are highlighted: additional Initialization and End-of-Query blocks; non-zero initial values; and calculation of the global maximum alignment score.

alignment locations) than others, and all alignments need to be kept in memory to be able to select the best overall alignment. Hence, in order to limit the hardware resources required for some extreme cases, not all reads are handled on the FPGA. In practice, we process more than 99% of all alignments on the FPGA. The remaining reads are instead executed on the host, which does not suffer from fixed memory size limitations.

B. Seed Extension Kernel

This section provides more details on the particular function of the Seed Extension kernel. Seeds, as generated by the SMEM Generation kernel, are an exact match of symbols from the read onto the reference (or a subsequence of either). The purpose of Seed Extension is to extend the length of such an exact match while allowing for small differences: mismatches between the read and reference, or skipping symbols on either the read or reference. A typical example of an alignment is given below:

```
                    Seed   Extension
Reference    GCGC AAGCTA GCTGAGGCTAA
Read         ---- AAGCTA AC-GAGG----
```

The Smith-Waterman algorithm [12] is a well-known dynamic programming algorithm that is guaranteed to find the optimum alignment between two sequences for a given scoring system. A *similarity matrix* is filled that computes the best score out of all combinations of matches, mismatches and gaps. This is illustrated by Figure 3. The process by which the similarity matrix is filled contains much inherent parallelism, as each cell only depends on its top, top-left and left neighbor. This implies that all cells on the same anti-diagonal can be computed in parallel.

1) Linear Systolic Arrays: A natural way to map dynamic programming algorithms onto reconfigurable hardware is as a linear systolic array. Many implementations that map the Smith-Waterman algorithm onto a systolic array have been proposed, amongst others [11], [13] and [14]. A systolic array consists of Processing Elements, or PEs for short, that operate in parallel. In the case at hand, we use such an array to take advantage of the available parallelism that exists while filling the similarity matrix, by processing the cells on the anti-diagonal in parallel. As illustrated in Figure 3, we map one read symbol to one PE, which corresponds to one column of the matrix. Each cycle, a PE processes one cell of the matrix and passes the resulting values to the next element. The values typically passed along to calculate the similarity matrix are the current cell's score, the row maximum, the current reference symbol, and the current gap score.

Although systolic array implementations excel in extracting parallelism, they do possess a number of drawbacks. First, the length of the PE-array determines the maximum length of a read that can be processed: one PE is required per read symbol. In this work we consider reads of up to 150 base pairs in length. Hence, we can guarantee that all reads will fit onto an array of a corresponding size.[3] Second, reads shorter than the PE-array still need to travel through it, incurring unnecessary latency and wasting resources by underutilizing the array. Finally, the maximum degree of parallelism is only achieved when all PEs are kept busy, which by virtue of its pipelined organization cannot always be ensured. In Section IV, we show how to deal with these issues.

[3]In practice, as we only consider data with a read length of 150 and the minimum seed length is 19 symbols, an extension can span at most 131 characters. Thus, an array of length 131 suffices.

2) Differences with "Standard" Smith-Waterman: The Seed Extension kernel used in BWA-MEM is similar to the Smith-Waterman algorithm, but the fact that the purpose is not to find the optimal match between two sequences, but instead to extend a seed found beforehand gives rise to three key differences. These differences and the impact they have on the design of the systolic array implementation are discussed below and illustrated in Figure 4.

1. Non-Zero Initial Values: Since the purpose of the Seed Extension kernel is to extend a seed, the match between sequences will always start from the "origin" of the similarity matrix (i.e., the top-left corner). The initial values provided to the first column and row of the dynamic programming matrix are not zero, but depend on the alignment score of the seed found by the SMEM Generation kernel.

Implication: An initial value block is placed in front of the array and initial values are computed and passed from one PE to the next.

2. Additional Output Generation: The Seed Extension kernel not only generates local and global alignment scores, which are the highest score in the matrix and the highest score that spans the entire read respectively, but also returns the exact location inside the similarity matrix where these scores have been found. Furthermore, a *maximum offset* is calculated that indicates the distance from the diagonal at which a maximum score has been found.

Implication: The index of the PE where the maximum is obtained is passed from one PE to the next. An End-of-Query block, which generates the output values by post-processing the results, is inserted at the end of the array.

3. Partial Similarity Matrix Calculation: To optimize for execution speed, BWA-MEM uses a heuristic that attempts to only calculate those cells that are expected to contribute to the final score. Profiling reveals that, in practice, only about 42% of all cells are calculated.

Implication: Our implementation does not use this heuristic, as the systolic array is able to perform all calculations on the anti-diagonal in parallel, which potentially leads to higher quality alignments.

IV. Design Space Exploration

Before deciding upon the final design of the Seed Extension kernel, a number of ideas and design alternatives, or *PE-module configurations*, were considered, varying in speedup, FPGA-resource consumption, suitability for certain data sets, and complexity. These are depicted in Figure 5 and will be discussed below. For analysis purposes, a data set with uniformly distributed extension lengths was used. Inspection of a histogram with GCAT seed extension lengths shows that this assumption is reasonable. We also consider that we have the entire data set available at the start for optimal scheduling.

A. Variable Logical Length Systolic Array

The length over which Seed Extension is to be performed is not the entire read length, but shorter, ranging from a single symbol up to the entire read length minus minimum seed length. Hence, the alignment that the kernel has to perform varies in its length. As mentioned in Section III-B1, a characteristic of systolic arrays is that processing time is independent of read length, as a read has to travel through the entire PE-array irrespective of its length: i.e., processing time is $O(|PEarray|+|Reference|)$, instead of $O(|Read|+|Reference|)$. Hence, shorter reads incur unnecessary latency and cause the systolic array to be underutilized.

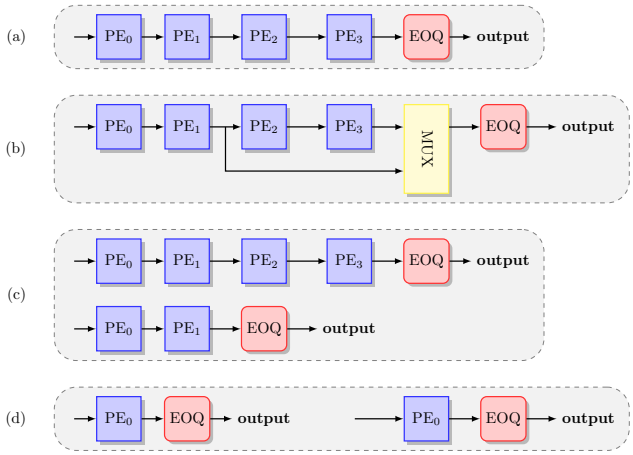

Fig. 5. PE-Module configurations: (a) standard systolic array configuration; (b) Variable Logical Length configuration that can bypass part of the array; (c) Variable Physical Length configuration that matches systolic array length to read length; (d) GPU-like single-PE modules.

To minimize latency, ideally a read would be processed by a PE-array matching its exact length. However, in practice, this is not achievable, since it would require having a PE-array for each possible read length, which is impractical given the available resources on the FPGA. Therefore, we propose to insert *multiple exit points* into the PE-array, as shown in Figure 5(b). We call this *Variable Logical Length* (VLL). This ensures that shorter reads do not have to travel through the entire array. Only a multiplexer and some control logic to select the correct exit point is needed, so this technique introduces minimal area overhead.

Definition 1 *The Exit Point Optimality measures how well an exit point configuration approximates the ideal situation of having a PE-array matching each read length.*

Of course, the Exit Point Optimality is data set dependent: for a set of reads that are all of the same length, a single module length suffices and the VLL technique can provide no benefit. In the case of a data set with uniformly distributed read lengths, it can be shown that minimal latency is achieved with evenly distributed exit points.

This idea can be further extended by subdividing the systolic array into two or more smaller logical systolic arrays that can operate in parallel. For example, a 150-PE array could present itself as two separate 75-PE arrays. This is similar to the approach suggested in [11]. However, that technique needs substantial additional hardware resources.

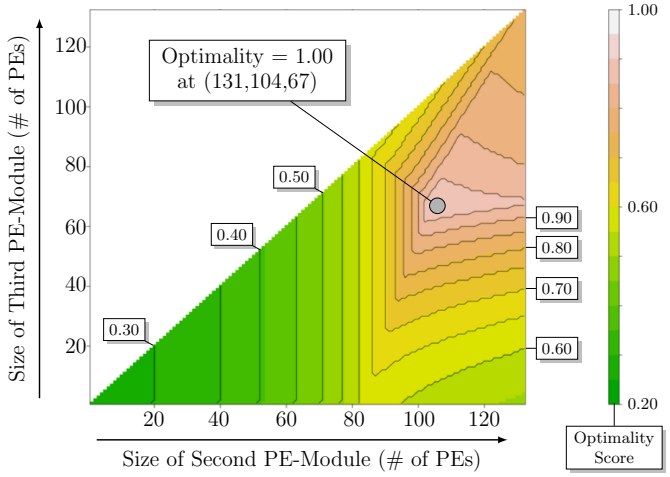

Fig. 6. Configuration Optimality for all three VPL PE-module combinations given a data set with uniformly distributed extension lengths. The indicated point shows the best configuration for the data set, which by definition has a value of 1. Data were obtained through exhaustive search of the design space.

B. Variable Physical Length Systolic Array

Another method to improve performance is using multiple systolic arrays together with *Variable Physical Length* (VPL), as shown in Figure 5(c). Longer reads are processed by the larger PE-arrays, while shorter reads are processed by the smaller arrays. Besides the above-mentioned improvement to execution speed, this has an additional benefit: the additional PE-arrays are physically smaller, which in turn allows for even more modules to be placed on the FPGA, improving speed even more. Combining this idea with the VLL-technique results in a *Variable Logical and Physical Length* (VLPL) array.

Definition 2 *The Configuration Optimality shows how closely a VPL-configuration achieves optimal load-balancing for a given data set, optimum CO being defined as 1.*

Figure 6 shows the Configuration Optimality for all combinations of three PE-modules, given a data set with uniformly distributed extension lengths. The graph shows the relative efficiency of all the configurations in the design space. To derive the relative efficiency, we have modeled the required time to process an entire data set, assuming optimal scheduling of reads onto modules. This is relatively easy given the fact that the processing time for each read is only dependent on the systolic array length of the modules and the length of the read to be processed. The optimal configuration is that configuration with the smallest module sizes (to minimize latency) for which all modules can be kept busy until the end.

Note that the optimal configuration of VLL- and VPL-arrays depends on the specific distribution of extension lengths of the data set at hand. In order to efficiently cope with various input data sets, multiple FPGA bitstreams can be compiled beforehand, each optimally configured for a different distribution. Then, an initial sampling of the input data can be used to select the best matching bitstream.

TABLE II
ESTIMATED RELATIVE PERFORMANCE OF SIMILAR AREA PE-MODULE CONFIGURATIONS GIVEN A DATA SET WITH UNIFORMLY DISTRIBUTED EXTENSION LENGTHS.

Design	PE-Module Configuration	Relative Speed	Relative Area
Standard	2x (131)	100%	100%
VLL	2x (131/87/43)[4]	125%	100%
VPL	1x (131), 1x (104), 1x (67)	181%	114%
VLPL	1x (131/122/113), 1x (104/92/79), 1x (67/45/22)	203%	114%

C. Single-PE Modules

The last alternative (see Figure 5(d)) is technically not a systolic array. One PE processes a similarity matrix entirely by itself. Parallelism is achieved not through intra-read parallelism, but instead by utilizing inter-read parallelism: processing multiple reads in parallel on a "sea-of-cores", similar to GPU-accelerated Smith-Waterman approaches such as the method discussed in [3]. Latency is traded for overall throughput.

An advantage of this approach is that it allows the use of the heuristic mentioned in Section III-B2-3, to only calculate relevant parts of the similarity matrix. Hence, as only about 42% of cells are processed, in theory a hundred single-PE modules should be 2.5 times faster than a one-hundred-PE module, not even considering the fact that the latter will often be underutilized. Drawbacks of this method are the considerable overhead this configuration suffers from: in PE-terms, control and other overhead are about equivalent to two PEs in logic cost. Moreover, whereas the systolic array designs implicitly store temporary data inside the array, the single-PE method requires explicit storage of temporary values. Finally, our current top level design can only fit up to six modules (of any kind), due to the resources required per core for input and output data structures.

D. Evaluation of PE-Module Configurations

Table II shows the relative performance of the various systolic array configurations. As we did not implement all the different configurations, we derived these estimations with the same approach as was used in Section IV-B to compute the Configuration Optimality. The configurations all have area requirements similar to a two 131-PE configuration. This will be the area on the FPGA we expect to be able to dedicate to seed extension logic in future implementations that also accelerate other parts of the algorithm on the FPGA. Given the extra resources the single PE configuration requires, we excluded this approach from the comparison.

The results show that the fastest approach is the VLPL configuration, being more than twice as fast as a standard systolic array implementation. The VLL and VPL configurations use different values for their exit points and module

[4]The Exit Point Optimality of this configuration is 0.87.

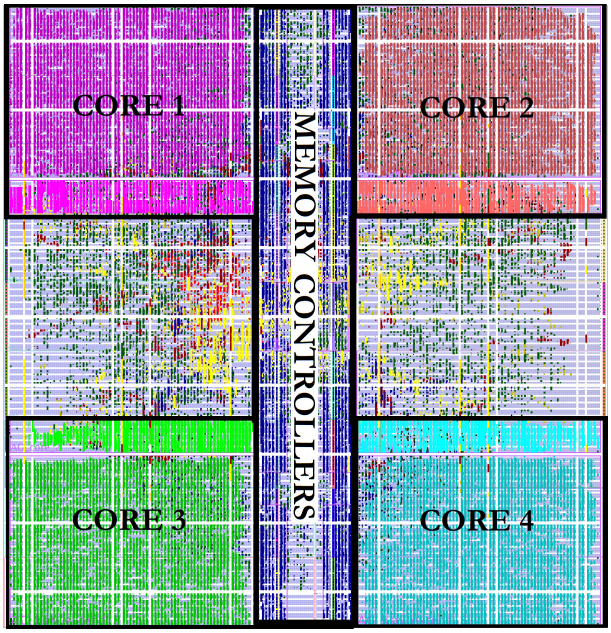

Fig. 7. Floorplan of the implemented FPGA design showing the four identical VLL-based modules and the memory controllers. The remainder of space is taken up by the Convey-to-host interface.

sizes, respectively, as their optimization goal is different: VLL optimizes for average latency, whereas VPL tries to balance execution time between modules. More exit points, or more modules would result in a higher speedup. The single-PE module configuration (not shown in the table) would be more than three times faster than a standard systolic array implementation, as (1) it does not suffer from underutilization of the array, and (2) it can take advantage of the similarity matrix heuristic (refer to Section III-B2-3).

V. Methods

All tests were run on the Convey HC-2^{EX} platform [2], configured with two Intel Xeon E5-2643 processors (four cores each, HyperThreading disabled) running at 3.3 GHz with 64 GB of DDR3 memory, paired with four Xilinx Virtex-6 LX760 FPGA co-processors (speed grade 1) connected to 64GB of SG-DIMM memory. Each FPGA is programmed with four Seed Extension modules, for a total of sixteen modules across all FPGAs. Modules are VLL-based and contain 131 PEs each.

To accelerate the Seed Extension kernel, we implemented a VLL-based design with four identical modules per FPGA, along with other components, such as memory controllers and the Convey-to-host interface blocks. Figure 7 shows the floorplan of the implemented design. A single module uses about 16% of FPGA resources, while in total approximately 71% of all resources was used. Although with more effort we would be able to place six modules per FPGA, a design with four modules provided sufficient performance to completely hide execution of the Seed Extension kernel. Hence, to reduce planning and routing complexity, we did not attempt to completely fill up the entire FPGA.

Data sets from the Genome Comparison & Analytic Testing (GCAT) framework [4] were used to obtain results for single-ended alignment (150bp-se-small-indel) and pair-ended alignment (150bp-pe-small-indel) of about eight million reads against the reference human genome (UCSC HG19). We used their online sequence alignment quality comparison service to verify that results of our FPGA-accelerated version are indistinguishable from those obtained with the BWA-MEM algorithm. We used BWA-MEM version 0.7.7 [6].

VI. Results

The results are summarized in Table III. Number of chunks indicates how many chunks are sent to the FPGA per batch. A value of one results in serial behavior, as then SMEM Generation and Seed Extension do not overlap. The last column shows the number of base pairs aligned per second.

A. Seed Extension Kernel

The table shows that the FPGA implementation of the Seed Extension kernel is up to three times faster than execution on the CPU, or 1.5 times faster when comparing a single module against one Xeon core. This implementation is fast enough to completely hide the execution of the Seed Extension kernel through overlapping its execution with SMEM Generation. Using a more advanced technique, such as VLPL, would allow us to achieve an even larger performance gain in Seed Extension, up to five-fold as compared to software-only execution (refer to Section IV-D). However, this would only benefit overall performance negligibly.

Note that the executions with only one chunk show slightly higher Seed Extension performance, due to less overhead from the chunking process. However, overall program execution time is lower, as no overlap between the two kernels is realized (refer to Figure 2 for more details).

B. Overall Program Execution

Offloading the Seed Extension kernel onto the FPGA results in an overall improvement to BWA-MEM execution time of up to 45%. Given the fact that BWA-MEM execution time is spread over three kernels (see Section II-B), we manage to attain 96% of the theoretically possible speedup of 47% from accelerating just this one kernel. Different numbers of chunks do not measurably impact performance, as long as overlap is possible between Seed Extension and SMEM Generation. A chunk size of one shows the isolated performance gain from Seed Extension acceleration without overlap. Note that BWA-MEM itself already offers multi-threaded execution.

VII. Discussion

By optimizing one of the three main BWA-MEM kernels, we realized an increase in application performance by up to 45%. Focusing on only one kernel leaves us exposed to the limitations clearly set out by Amdahl's law, limiting the potential gains in performance. Our next efforts are hence focused on accelerating the other kernels.

The Seed Extension kernel proved to be a good fit to port to the FPGA, although it is obvious that even just porting one

TABLE III
EXECUTION TIME AND SPEEDUP FOR THE GCAT ALIGNMENT QUALITY BENCHMARK

			Seed Extension Kernel		Overall Program		
Test	**Platform**	**# of Chunks**	**Execution Time**	**Speedup**	**Execution Time**	**Speedup**	**Throughput**
Single-Ended	CPU only	-	167 s	-	530 s	-	2.3 Mbp/s
Data	CPU+FPGA	11	62 s	2.69x	366 s	1.45x	3.3 Mbp/s
	CPU+FPGA	6	62 s	2.70x	365 s	1.45x	3.3 Mbp/s
	CPU+FPGA	1	61 s	2.73x	412 s	1.29x	2.9 Mbp/s
Pair-Ended	CPU only	-	172 s	-	545 s	-	2.2 Mbp/s
Data	CPU+FPGA	11	63 s	2.75x	402 s	1.35x	3.0 Mbp/s
	CPU+FPGA	6	62 s	2.78x	400 s	1.36x	3.0 Mbp/s
	CPU+FPGA	1	61 s	2.82x	447 s	1.22x	2.7 Mbp/s

kernel has wider implications to the program structure than just replacing a single function call: limitations in memory transfer efficiency forced us to reorder the program execution into batches to allow for larger, more efficient data transfers. Moreover, the acceleration potential of using FPGAs is largely dependent on data size. The huge parallelism an FPGA can offer, granting $O(n)$ scaling as compared to $O(n^2)$ on the host CPU, will become much more apparent at longer read sizes: a read length of 1,000 symbols would result in a ten-fold speedup, compared to the 1.5x speedup we managed to attain. Hence, it is important to have a deep understanding of the data set at hand before applying a general solution. Finally, knowledge of the extension length distribution is required to implement a PE-module design with optimal efficiency.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have presented the initial results of our efforts to accelerate BWA-MEM. We propose the first accelerated version of BWA-MEM, offloading one of the three main program kernels onto an FPGA and overlapping its execution. We implemented the Seed Extension kernel as a systolic array and achieved performance for this kernel up to three times faster than software-only execution. This translates into an overall improvement to execution time up to 45%, close to the theoretical maximum of 47%, as the kernel's execution time is almost completely hidden. Moreover, we have presented generally applicable techniques to improve the performance of variable length Smith-Waterman systolic arrays by up to three times, with very little area overhead.

Our next efforts will focus on offloading the other kernels of the BWA-MEM algorithm onto the FPGA, for which SMEM Generation is a natural candidate. We will also investigate the implementation of a VLPL-module, mostly as area savings measure, as the gains in speed can be used to reduce the allocated space of the kernel on the FPGA. Successful acceleration of BWA-MEM will bring us one step closer to overcoming the computational bottlenecks inherent in the Next Generation Sequencing genomics pipeline.

IX. ACKNOWLEDGMENTS

The authors would like to thank Bluebee and Convey Computer for kindly making available the resources for testing and implementation work, and in particular thank Giacomo Marchiori from Bluebee for all his efforts.

REFERENCES

[1] H. Cao, Y. Wang, W. Zhang, X. Chai, X. Zhang, S. Chen, F. Yang, C. Zhang, Y. Guo, Y. Liu, et al. A short-read multiplex sequencing method for reliable, cost-effective and high-throughput genotyping in large-scale studies. *Human mutation*, 34(12):1715–1720, 2013.

[2] Convey Computer. The Convey HC-2 architectural overview. http://www.conveycomputer.com. Accessed: 2014-11-04.

[3] L. Hasan, M. Kentie, and Z. Al-Ars. DOPA: GPU-based protein alignment using database and memory access optimizations. *BMC research notes*, 4(1):261, 2011.

[4] G. Highnam, J. J. Wang, D. Kusler, J. Zook, V. Vijayan, N. Leibovich, and D. Mittelman. An analytical framework for optimizing variant discovery from personal genomes. *Nature communications*, 6, 2015.

[5] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[6] H. Li. Burrows-Wheeler Aligner. http://bio-bwa.sourceforge.net/. Accessed: 2014-11-04.

[7] H. Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.

[8] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[9] H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851–1858, 2008.

[10] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[11] T. Oliver, B. Schmidt, and D. Maskell. Hyper custo- mized processors for bio-sequence database scanning on FPGAs. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 229–237. ACM, 2005.

[12] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[13] C. W. Yu, K. Kwong, K.-H. Lee, and P. H. W. Leong. A Smith-Waterman systolic cell. In *New Algorithms, Architectures and Applications for Reconfigurable Computing*, pages 291–300. Springer, 2005.

[14] P. Zhang, G. Tan, and G. R. Gao. Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform. In *Proceedings of the 1st international workshop on High-performance reconfi- gurable computing technology and applications: held in conjunction with SC07*, pages 39–48. ACM, 2007.