An Efficient GPU-Accelerated Implementation of Genomic Short Read Mapping with BWA-MEM

Ernst Joachim Houtgast[1,2], Vlad-Mihai Sima[2], Koen Bertels[1] and Zaid Al-Ars[1]
[1] Computer Engineering Lab, TU Delft, Mekelweg 4, 2628 CD Delft, The Netherlands
[2] Bluebee, Laan van Zuid Hoorn 57, 2289 DC Rijswijk, The Netherlands
Corresponding author: ernst.houtgast@bluebee.com

ABSTRACT

Next Generation Sequencing techniques have resulted in an exponential growth in the generation of genetics data, the amount of which will soon rival, if not overtake, other Big Data fields, such as astronomy and streaming video services. To become useful, this data requires processing by a complex pipeline of algorithms, taking multiple days even on large clusters. The mapping stage of such genomics pipelines, which maps the short reads onto a reference genome, takes up a significant portion of execution time. BWA-MEM is the de-facto industry-standard for the mapping stage.

Here, a GPU-accelerated implementation of BWA-MEM is proposed. The Seed Extension phase, one of the three main BWA-MEM algorithm phases that requires between 30%-50% of overall processing time, is offloaded onto the GPU. A thorough design space analysis is presented for an optimized mapping of this phase onto the GPU. The resulting systolic-array based implementation obtains a two-fold overall application-level speedup, which is the maximum theoretically achievable speedup. Moreover, this speedup is sustained for systems with up to twenty-two logical cores. Based on the findings, a number of suggestions are made to improve GPU architecture, resulting in potentially greatly increased performance for bioinformatics-class algorithms.

1. INTRODUCTION

The introduction of Next Generation Sequencing (NGS) techniques has resulted in drastic, ongoing, cost reduction of genomic sequencing, which, in turn, has led to an enormous growth in the amount of genetic DNA data that is being sequenced. High-throughput sequencing facilities are coming online around the world as facilities worldwide embrace NGS [2]. The amount of data being generated is projected to rival, if not outright overtake, other key Big Data-fields, such as astronomy and streaming video services [13].

NGS machines output so-called *short reads*, short fragments of DNA of at most a few hundred base pairs (bp) in length. This data requires extensive processing using a *genomics pipeline*, which typically contain multiple stages with a number of highly complex algorithms. In the case of a DNA sequencing pipeline, first, the millions of short reads generated are mapped onto a reference genome. Then, these mapped reads are sorted and duplicates are marked or removed. Finally, the aligned data is compared at several positions with known possibilities, in order to determine the most probable variant. Only then the data is ready for consumption by the end-user, such as a clinician or researcher. These variants, or mutations, are generally what is of interest, as such a mutation could give insight on which is the most effective treatment to follow for the particular illness a patient has. The mapping stage takes a significant portion of processing time for a typical pipeline execution, around 30%-40%, depending on data set and platform.

A sequencing run on an Illumina HiSeq X, a state-of-the-art NGS sequencer, produces data in the order of 450 GB. For cancer data sets, this data requires multiple days of processing, even on high performance computing clusters. The extreme scale of data and processing requires enormous computing capabilities to make the analysis feasible within a realistic time frame. As heterogeneous computing holds great potential to provide large advantages in speed and efficiency, in this paper, we demonstrate the effectiveness of GPU-based acceleration of BWA-MEM, the most widely used tool for the mapping stage of genomics pipelines.

The following contributions are made: 1) an optimized GPU-based implementation of the BWA-MEM Seed Extension phase, resulting in an overall application-level speedup of up to 2x; 2) a thorough discussion of the design space analysis, providing key insight into the requirements of a highly optimized implementation; and 3) recommendations to further improve the GPU architecture that would allow even higher performance for bioinformatics-class applications.

The remainder of this paper is organized as follows. In Section 2, related work is discussed. In Section 3, background information is given on the BWA-MEM algorithm and, in particular, on the Seed Extension phase. In Section 4, the advantages and disadvantages of various implementation architectures are reviewed. In Section 5, the results are presented. Section 6 contains a discussion of the results and recommendations are made to improve the GPU architecture. Section 7 concludes the paper.

2. RELATED WORK

Numerous GPU-accelerated implementations of short read mapping tools exist, notable examples include SOAPv3 [9] and CUSHAW [10]. Similar to BWA-MEM and most other state-of-the-art mapping tools, these consist of an Exact Matching phase using the Burrows-Wheeler transform to find exactly matching subsequences, followed by an Inexact Matching phase. However, these implementations are limited in the flexibility of their Inexact Matching algorithm, allowing only for a small number of mismatches (CUSHAW), or by disallowing gaps in the alignment (SOAPv3).

Using a variant of the Smith-Waterman (SW) algorithm [12] for its Inexact Matching, BWA-MEM does not impose such limitations. For example, gaps do not influence performance. The SW algorithm is a dynamic programming technique able to find the optimal match between two subsequences given a certain scoring scheme. Many accelera-

To appear in the International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, July 2016, Hong Kong.

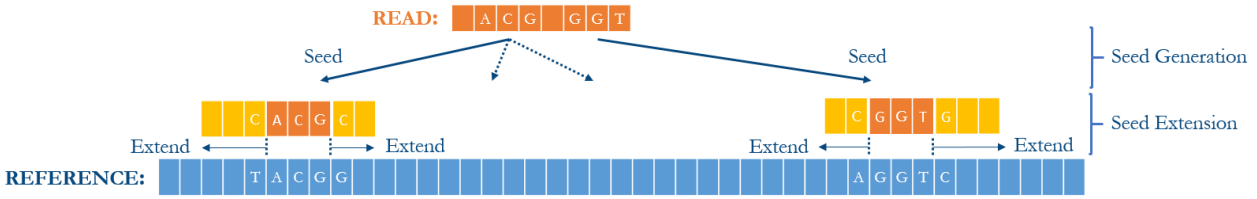

Figure 1: BWA-MEM processes reads using the Seed-and-Extend paradigm: for each read, likely mapping locations on the reference are found by searching for *seeds*, exactly matching subsequences between the read and the reference. These seeds are then extended in both directions using a Smith-Waterman-like approach allowing for inexact matches. The best scoring alignment is selected.

ted implementations of this algorithm exist (e.g., [8], [11]). However, all these implementations perform one complete sequence matching per compute thread, making such an implementation unsuitable for direct application onto BWA-MEM, as it requires batching and sorting of larger groups of work. Section 3.2 explains in more detail why such a parallelization strategy is inapplicable for BWA-MEM.

To the authors' knowledge, only a few accelerated implementations of BWA-MEM exist: two FPGA implementations of BWA-MEM on the Convey supercomputing platform: one offloading the Seed Extension phase onto four Xilinx Virtex-6 FPGAs [4] obtaining a 1.5x speedup, the other accelerating multiple BWA-MEM phases [1] obtaining a 2.6x speedup; and a GPU-accelerated implementation of the Seed Extension phase [5], achieving a 1.6x speedup. This work improves upon [5], obtaining far better results: a two-fold speedup for a system with up to twenty-two logical cores is obtained, compared to an at most 1.6x speedup for a system with up to four cores. Moreover, an NVIDIA GeForce GTX 970 is used, compared to using a setup with dual NVIDIA GeForce GTX TITAN X, equivalent to about one-third of the GPU resources. Note that all these implementations are actual production-quality implementations.

3. BACKGROUND

There are a number of traits that bioinformatics-class algorithms share, making them interesting, but nevertheless challenging candidates for acceleration efforts. The two most important ones are outlined below:

Extreme-Scale Data Size: The data size that many bioinformatics applications deal with are of an enormous magnitude, for example illustrated in the case of NGS sequencing. A single human genome contains three billion base pairs. A base is one out of four possible nucleotides (A, C, G or T). Moreover, the sequencer also provides a quality score for each base, which indicates the confidence with which the nucleotide was read. Finally, as only short fragments are sequenced and this data often contains errors, it is common practice to read the genome multiple times, a *coverage* of 30x or more being typical. This results in a compressed output size of around 100 GB.

Often, this huge amount of data coincides with an abundance of parallelism. For example, in the case of BWA-MEM, short reads can be mapped in parallel, as there exist no dependencies between them.

Complex Multikernel Algorithms: Typical bioinformatics algorithms do not consist of a single phase that dominates execution time, but instead perform a number of time-consuming steps. For example, BWA-MEM processing is spread over three distinct stages, making acceleration of this algorithm more challenging, as not only does it require the adaptation of multiple separate algorithms, but also care has to be taken to not shift the bottleneck to another part of the application, limiting the benefit of any potential speedup as per Amdahl's law. This makes it quite difficult to obtain larger performance gains.

3.1 The BWA-MEM Algorithm

The goal of the BWA-MEM algorithm is to find the best mapping of a short read onto a reference genome [7]. To achieve this, it makes use of the Seed-and-Extend paradigm (refer to Figure 1), a two-step method consisting of an Exact Matching phase and an Inexact Matching phase (for details, see [1]). First, for each short read Seed Generation is performed: exactly matching subsequences of the read and reference called *seeds* are identified using a Burrows-Wheeler Transform-based index. The BWT-method allows for efficient string-lookup and forms the fundament of almost all contemporary state-of-the-art mapping tools. A single short read can have many such seed locations identified. Generated seeds that are found to be in close proximity of each other on the reference genome are grouped into *chains*.

The Seed Generation phase is followed by a Seed Extension phase. Here, seeds found earlier on are *extended* using an algorithm similar to the widely-used Smith-Waterman algorithm, using a scoring system that awards matches and penalizes mismatches, insertions and gaps. Typically, not all seeds are extended. Instead, on average only one seed per chain is extended. Out of all the extended seeds, the highest scoring match is chosen as final *alignment*.

3.2 Seed Extension Phase

BWA-MEM contains three main computational phases: Seed Generation, Seed Extension and Output Generation. During Output Generation the best alignment is selected and the output is written. Seed Extension typically requires between 30%-50% of execution time [5]. As per Amdahl's law, the maximum obtainable speedup for only accelerating this phase is thus limited to a two-fold speedup at best.

This paper focuses on GPU-based acceleration of the Seed Extension phase, which consists of two main parts: an outer loop that loops over all the seeds identified for the read during Seed Generation, and an Inexact Matching kernel, which performs the Smith-Waterman-like extension.

There are no dependencies between reads and thus reads can be processed in parallel. For each read, the groups of chains are processed iteratively, as the check for overlap between earlier found alignment regions introduces a dependency in the program order. This dependency is the main reason why typical Smith-Waterman GPU-implementations are not applicable for the case of BWA-MEM: they ob-

tain their speed by performing many Smith-Waterman alignments in parallel, which are batched and sorted together in larger groups of approximately the same length for load balancing purposes. Due to the highly dynamic nature of the Inexact Matching invocations, this is impractical to achieve for BWA-MEM, and would at the very least require a major algorithm overhaul, if at all possible.

3.3 Inexact Matching Kernel

The Inexact Matching algorithm is similar to the popular Smith-Waterman dynamic programming algorithm, which computes for a given scoring scheme the optimal alignment between two subsequences by filling a similarity matrix, resulting in a maximum score. Backtracking can be used to obtain the actual path through the similarity matrix that results in the optimal alignment. However, the algorithm is computationally expensive, being of $O(read \times reference)$. Therefore, most mapping tools use an initial Seeding-phase to find likely mapping locations, and only then perform localized extension of these seeds.

The Inexact Matching algorithm of BWA-MEM is slightly different from regular Smith-Waterman. Firstly, since the algorithm is used to extend a seed, the initial values used in the similarity matrix are non-zero. Secondly, some additional outputs are generated, most importantly the location of the maximum in the similarity matrix, and a global maximum with its location. Since each value in the similarity matrix only depends on its top, left, and top-left neighbor, the anti-diagonals of the similarity matrix can be computed in parallel, thus making a systolic array a natural implementation approach. Each column of the similarity matrix is processed in parallel by a Processing Element (PE) of the systolic array, thus reducing the processing time from $O(read \times reference)$ to $O(read + reference)$. This results in speedups of several orders of magnitude for longer read and reference sequences. However, as BWA-MEM is typically used for shorter reads of at most a few hundred base pairs, the observed speedup is more modest.

4. DESIGN SPACE EXPLORATION

As explained in Section 3.2, the BWA-MEM Seed Extension phase consists of two very distinct parts: the Inexact Matching algorithm, which is implemented as a systolic array, and the Seed Extension main loop, that loops over all the chains of seeds. This outer loop performs the sequential tasks of control and branch operations to effectuate the looping over all seeds, proper decoding of the sequence and reference from main memory, and writes the result back to memory. In contrast, the Inexact Matching function is highly computationally intensive and can use as many threads as the systolic array has PEs. Thus, the implementation in [5] on which this work is based makes a clear separation between both functions and utilizes CUDA Dynamic Parallelism to dynamically instantiate Inexact Matching kernels as needed. A number of kernels were implemented, each optimized for different matrix dimensions, and are called appropriately. However, our tests show that CUDA Dynamic Parallelism brings about a large initialization penalty, making it unsuitable to use at this scale, where even a single read can generate thousands of calls, resulting in millions of invocations during a typical program execution. Therefore, the implementation here does not make use of Dynamic Parallelism, and instead uses a large monolithic kernel.

4.1 GPU-Based Inexact Matching

The main challenge of the GPU-accelerated Seed Extension function is the implementation of the Smith-Waterman-like Inexact Matching kernel. Typical GPU implementations of Smith-Waterman extract parallelism by performing many sequence alignments in parallel. Here, parallelism is extracted from an individual alignment by harnessing the parallelism residing in the anti-diagonals of the similarity matrix, through use of a systolic array (see Figure 2).

The *warp* is the basic unit of action on an NVIDIA GPU. All threads in a warp perform the same operation, and jobs are always scheduled onto one or more complete warps. Therefore, two types of systolic array implementations were considered. A "wide" systolic array implementation, mapping each PE onto a separate thread and using as many warps as needed, and a single warp implementation, using only a single warp and processes the similarity matrix in multiple passes. These approaches differ in their utilization of Shared Memory (SM), a limited on-chip resource. The amount of SM a thread block requires directly puts an upper bound on the number of thread blocks that can be resident, thus impacting performance.

As data flows through the systolic array, the PEs exchange data with their neighbors to share results and perform their computations. The "wide" implementation uses SM to simulate the data exchange between PEs. After computation of each antidiagonal, each PE passes its results to the next PE in the array. Thus, the amount of SM required depends on the length of the systolic array, which in turn depends on the length of the read. Explicit synchronization between warps is required after each step, which can be costly.

In contrast, the single warp implementation requires storage for the data produced at the "border" of each pass, which is fed back into the array during the next pass. Therefore, the amount of SM required depends on the length of the reference query. A large advantage of using a single warp is that intra-warp shuffle instructions can be used,

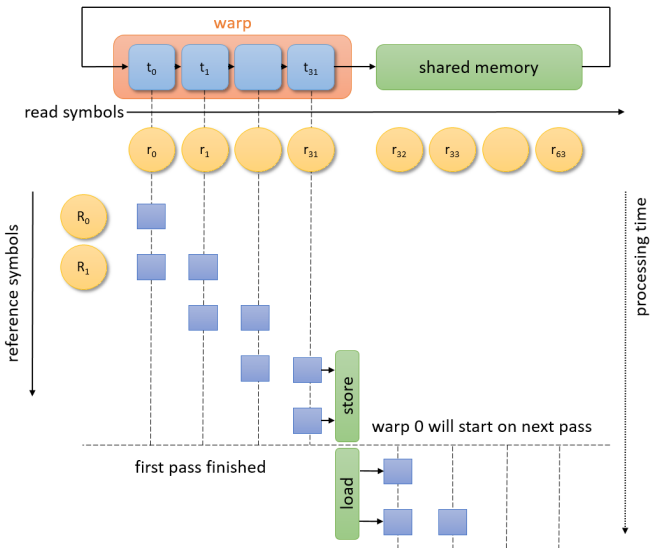

Figure 2: Read symbols map onto the systolic array of threads. Multiple passes are required to process all read symbols. Data exchange between passes is implemented using Shared Memory.

which allow threads within one warp to directly access each other's registers. This eliminates the need for data exchange through SM, saving a huge amount of bandwidth. In this approach, only the first PE in the warp needs to read, and the last PE needs to write temporary data. Another advantage is that intra-thread synchronization within one warp is cheaper. Compared to the "wide" implementation, the single-warp implementation has a secondary benefit as it eliminates a large drawback of systolic arrays: the difficulty of keeping all PEs busy. Depending on the similarity matrix dimensions, many of the PEs can be idle much of the time. With the single warp implementation, this inefficiency is drastically reduced by skipping those parts of the matrix where all the PEs of the pass would be idle.

To implement the Smith-Waterman algorithm, each CUDA thread performs the pseudo code as shown in Algorithm 1: each pass, the warp of threads is assigned a new part of the read symbols to process. Then, all the calculations are performed for this subsection of the similarity matrix. Each cycle, each thread reads its left neighbor's results, this way implementing the systolic array behavior.

4.2 Implementation Architecture

Due to the above reasons, the single-warp systolic array design is implemented. To maximize occupancy, Shared Memory and register usage was carefully balanced. The register count was fixed to use 64 registers per thread. The maximum number of storage between passes was chosen such to ensure that one thread block uses 2 kB of Shared Memory. Hence, up to 32 thread blocks can be resident per multiprocessor. Analysis performed with the NVIDIA Visual Profiler shows that the performance is mostly limited by latency of arithmetic and memory instructions. The memory subsystem is not very much utilized, as the Shared Memory bandwidth is only 158 GB/s and the device memory bandwidth is less than 5 GB/s. The GPU caching is effective, as device memory bandwidth is substantially lower than overall unified cache bandwidth.

Our approach is limited by the fact that the latest NVIDIA GPU architectures (Compute Capability 5.0) can have up to 2048 resident threads active per multiprocessor, but only 32 blocks. For optimal occupancy, thread blocks with at least 64 threads should be used, whereas here only 32 threads are used per block. Hence, occupancy is limited to at most 50%. In practice, up to about 35% occupancy is realized. Earlier Compute Capability versions were even more restrictive, only allowing 16 resident blocks per multiprocessor for Compute Capability 3.0+, or just 8 resident blocks per multiprocessor for earlier architectures. This would have a direct impact on the efficiency of this implementation.

Algorithm 1 Systolic Array CUDA Thread Pseudo Code

```
1: for (each pass) do
2:    Load current read symbol
3:    for (each reference symbol + warp size) do
4:       if (active) then
5:          Load left neighbor values
6:          Perform Seed Extension cell computations
7:       end if
8:    end for
9: end for
```

4.3 BWA-MEM Optimizations

A number of optimizations has been implemented, resulting in much better performance as compared to the GPU-based implementation in [5]:

Single Monolithic Kernel: Although in theory Dynamic Parallelism should help improve occupancy by lowering resource requirements, the incurred performance overhead makes it unfeasible to use when it needs to instantiate kernels dynamically on such an extremely large scale.

Memory Subsystem Optimizations: GPUs contain specialized memory subsystems. The reference and input data are placed inside read-only texture memory to take advantage of locality. Constants are used for parameters such as scoring variables to reduce register count.

Truncated Reference Length: Analysis of the Seed Extension algorithm shows that it is unnecessary to process the part of the similarity matrix where the reference is much longer than the read, given that this would imply numerous gaps or insertions, and thus a low score. The highest score will be found in the upper part of the similarity matrix.

5. EXPERIMENTAL RESULTS

The optimized GPU implementation described here was tested on a system with an Intel Core i7-4790 (3.6 GHz, eight logical cores), SpeedStep and Hyper-Threading enabled, containing 16 GB of DDR3 memory, and an NVIDIA GeForce GTX 970 with 1664 CUDA cores and 4 GB of on-board RAM. CUDA version 7.5 was utilized. Results for the GPU implementation described in [5] were obtained on a system with an Intel Core i7-4790 (4.0 GHz, eight logical cores), SpeedStep and Hyper-Threading enabled, containing 32 GB of DDR3 memory, and two NVIDIA GeForce GTX TITAN X cards, with 3,072 CUDA cores each. The FPGA results were obtained on a system with an Intel Core i7-4790 (3.6 GHz, eight logical cores), SpeedStep and Hyper-Threading enabled, containing 16 GB of RAM and a server-grade Alpha Data ADM-PCIE-7V3 card with a Xilinx Virtex-7 XC7VX690T-2 and 16 GB of on-board RAM, programmed with six Seed Extension modules at 160 MHz [6].

BWA-MEM version 0.7.8 was used with publicly available data sets for single-ended alignment (150bp-se-small-indel) and pair-ended alignment (150bp-pe-large-indel) from the Genome Comparison & Analytic Testing (GCAT) framework [3]. These contain about eight million reads of 150 base pairs, about 1.2 billion base pairs in total. Reads were aligned against the reference human genome (UCSC HG19).

5.1 Performance Results

Performance results are summarized in Table 1, given as execution time in seconds as well as in throughput in millions of base pairs per second. This facilitates cross-data set and cross-platform comparisons. To distinguish the GPU implementations, the implementation from [5] is referred to as *GPU-accelerated*, and the implementation proposed here is called *GPU-optimized*. These two implementations are compared to the FPGA-implementation from [6], which uses the server-grade Alpha Data add-in card. The time for the Seed Extension phase is omitted for the GPU-accelerated implementation, as the number reported there is not directly comparable, as it only includes the Inexact Matching computation time, and not the time required to process the outer loop. Moreover, the GPU-optimized implementation described here processes almost all reads, whereas the GPU-

Table 1: Execution time and speedup for the synthetic GCAT alignment quality benchmark

		Seed Extension Phase		Overall Application		
Test	**Platform**	**Execution Time**	**Speedup**	**Execution Time**	**Speedup**	**Throughput**
Single-Ended Data	Software-Only	237 s	-	552 s	-	2.2 Mbp/s
	FPGA-Accelerated [6]	129 s	1.8x	272 s	2.0x	4.5 Mbp/s
	GPU-Optimized	144 s	1.6x	278 s	2.0x	4.3 Mbp/s
	Software-Only [5]	218 s	-	510 s	-	2.4 Mbp/s
	GPU-Accelerated [5]	N/A	N/A	422 s	1.2x	2.9 Mbp/s
Pair-Ended Data	Software-Only	246 s	-	572 s	-	2.1 Mbp/s
	FPGA-Accelerated [6]	130 s	1.9x	289 s	2.0x	4.1 Mbp/s
	GPU-Optimized	141 s	1.7x	293 s	2.0x	4.1 Mbp/s

accelerated implementation is only able to process about 99.5% of all reads, leaving the most time-consuming reads for the host CPU. The results for the two software-only platforms differ slightly, as their clock frequency is slightly different (4.0 GHz vs 3.6 GHz). Both the GPU-optimized implementation and the FPGA-accelerated implementation are able to reach a 2x speedup, compared to software-only execution. The GPU-accelerated implementation only reaches a 1.2x speedup. The results for the GPU-optimized implementation are much better than for the GPU-accelerated implementation, even though a GPU-subsystem is used with only about 31% of the computational resources.

Note that we deliberately refrain from making a direct comparison between BWA-MEM and other read mapping tools, as in this field, strict reproducibility is critical, making performance of other tools irrelevant.

5.2 Scalability Analysis

Besides overall performance, scalability of the implementations is also important: the number of CPU cores a system can have for which the system is still accelerated with maximum speedup. This is estimated by considering the time required by the accelerator for the Seed Extension phase and regarding this as a lower bound to overall execution time. Assuming overall execution time scales linearly in processor core count, which for BWA-MEM is not unreasonable, the maximum number of logical CPU cores that can be effectively accelerated is thus determined. The results are summarized in Table 2. Results are also included for further optimized implementations that include certain straightforward improvements to scalability behavior, such as further decomposition and pipelining of the data preprocessing step that prepares the data for the GPU or FPGA, indicated as *FPGA-opt* and *GPU-opt* in the table.

The scalability results are also visually depicted in Figure 3, showing speedup compared to a host system with the same number of cores. The increased speedup when using eight threads may be caused by Hyper-Threading, which makes Seed Extension a larger part of overall execution time due to being the least memory-intensive phase, thus benefitting the least from Hyper-Threading.

Table 2: Scalability Analysis of the Implementations

	Execution Time			
Platform	**Seed Ext.**	**Overall**	**Utilization**	**Scalability**
FPGA	129 s	272 s	47%	16.8 cores
FPGA-opt	83 s	272 s	31%	26.1 cores
GPU	144 s	278 s	52%	15.4 cores
GPU-opt	101 s	282 s	36%	22.3 cores

6. DISCUSSION

This section presents lessons learned during the implementation work, and gives some recommendations to improve GPU architecture for bioinformatics-class problems.

6.1 Lessons Learned

The abundance of parallelism in BWA-MEM, and many bioinformatics-class algorithms in general, makes them interesting candidates for GPU-based acceleration. However, the complexity of these algorithms, with execution time distributed over multiple distinct phases, makes obtaining large overall application-level performance gains far from straightforward. Bottlenecks quickly shift towards the non-accelerated parts of the program.

For BWA-MEM Seed Extension, the existence of an outer loop dynamically calling Inexact Matching functions makes it ill-suited to apply typical Smith-Waterman acceleration methods. Therefore, a systolic array approach was used to accelerate the algorithm instead. To avoid large temporary storage requirements for data transfer between systolic array PEs, only a single warp was used, allowing the use of intra-warp shuffle. This greatly reduces Shared Memory bandwidth requirements. However, it puts an upper limit on the achievable occupancy, as only 32 threads are instantiated per thread block, whereas optimal occupancy can only be obtained with at least 64 threads per thread block.

Dynamic parallelism, as used in [5], seems like a valid approach given the two distinct parts of the Seed Extension algorithm. However, in practice, it brings about a large overhead. This shows the importance of testing a wide range of implementations, and not just choosing the approach that in theory should be the best.

6.2 Recommended Architecture Optimizations

The lessons learned during implementation of the GPU-based Seed Extension phase have given key insights in the requirements of bioinformatics-class algorithms. Therefore, here follow a number of suggestions to GPU architecture that could greatly improve performance for such algorithms:

Reduced Dynamic Parallelism Overhead: Reduced Dynamic Parallelism overhead could make this into a valid approach to reduce register and Shared Memory pressure, thus improving occupancy.

Increased Resident Blocks per Multiprocessor: The present limit of 32 resident blocks per multiprocessor limits the maximum obtainable occupancy for single warp thread blocks. Raising this limit to 64 resident blocks per multiprocessor could result in an up to 100% boost to performance, as this limitation is the major obstacle to higher performance in

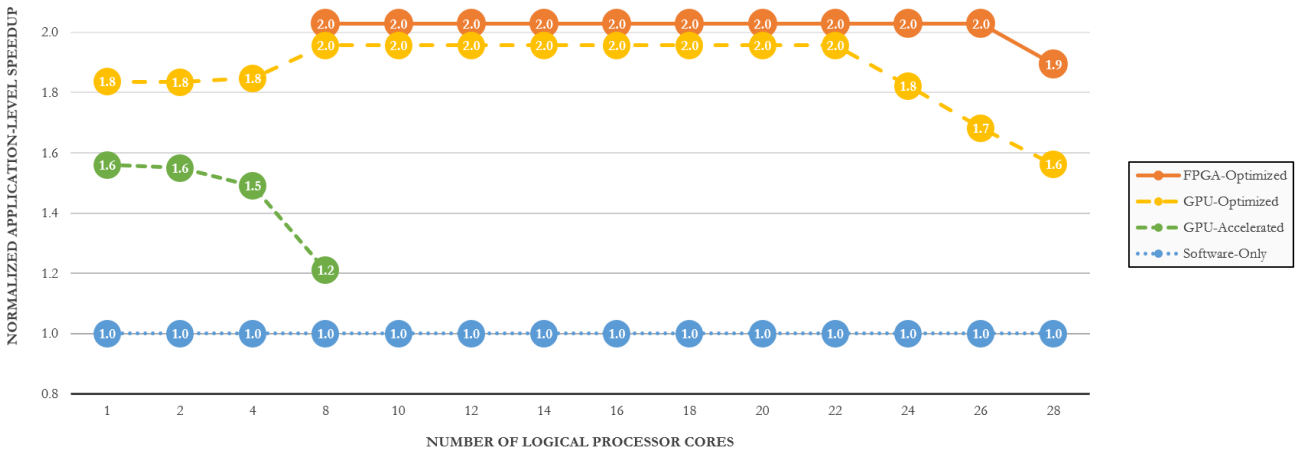

Figure 3: Estimated application-level speedup compared to software-only execution on a system with identical number of CPU cores. The GPU-Optimized implementation is able to sustain a two-fold speedup for systems with up to twenty-two CPU cores, greatly exceeding the results of the GPU-Accelerated implementation, which uses dual NVIDIA GeForce GTX TITAN X compared to the single GeForce GTX 970 here.

this work. Not only the implementation here would benefit, but also other Smith-Waterman implementations, a staple algorithm in bioinformatics. In general, this would improve any implementation that relies on the intra-warp shuffle capability and are thus limited to a single warp.

Native Low Precision Data Formats: Many problems in bioinformatics do not require high precision for their calculations. For example, the maximum value of entries in the Smith-Waterman similarity matrix can be easily determined based on the scoring parameters and length of both sequences. In many instances, even eight bits of precision is sufficient. The current native 32-bits of precision minimum results in wasted register and Shared Memory space, unless tricks are performed that require additional instructions.

7. CONCLUSIONS

In this paper, a GPU-accelerated implementation is described of the BWA-MEM genomic mapping algorithm. The Seed Extension phase is one of the three main BWA-MEM program phases, which requires between 30%-50% of overall execution time. Offloading this phase onto the GPU provides an up to twofold speedup in overall application-level performance. Analysis shows that this implementation is able to sustain this maximum speedup for a system with at most twenty-two logical cores. This can save days of processing time on the enormous real-world data sets that are typical of NGS sequencing.

The implementation presented here greatly exceeds the performance of the GPU implementation of [5], offering a higher speedup of 2x for systems with up to twenty-two cores, compared to 1.6x for systems with up to four cores, even while at the same time using a GPU-subsystem that only provides about 31% of the computational capabilities.

Although the work here focuses on BWA-MEM, a widely used genomic mapping tool, the approach used is valid for many similar Seed-and-Extend-based bioinformatics algorithms. Moreover, based on the insights obtained, a number of optimizations to GPU architecture are suggested: reduced Dynamic Parallelism overhead, increased number of resident blocks per multiprocessor, and native low precision data formats. These should greatly improve GPU performance for bioinformatics-class problems.

8. REFERENCES

[1] N Ahmed, V Sima, EJ Houtgast, KLM Bertels, and Z Al-Ars. Heterogeneous Hardware/Software Acceleration of the BWA-MEM DNA Alignment Algorithm. In *Proc. of the IEEE/ACM Intl. Conf. on Computer-Aided Design*, ICCAD, 2015.

[2] James Hadfield and Nick Loman. Next Generation Genomics: World Map of High-throughput Sequencers. `http://omicsmaps.com`, 2016. Accessed: 2016-01-13.

[3] Gareth Highnam, Jason J Wang, Dean Kusler, Justin Zook, Vinaya Vijayan, Nir Leibovich, and David Mittelman. An Analytical Framework for Optimizing Variant Discovery from Personal Genomes. *Nature comm.*, 6, 2015.

[4] EJ Houtgast, V Sima, KLM Bertels, and Z Al-Ars. An FPGA-Based Systolic Array to Accelerate the BWA-MEM Genomic Mapping Algorithm. In *Intl. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2015.

[5] EJ Houtgast, V Sima, KLM Bertels, and Z Al-Ars. GPU-Accelerated BWA-MEM Genomic Mapping Algorithm Using Adaptive Load Balancing. In *Architecture of Computing Systems–ARCS*, pages 130–142. Springer, 2016.

[6] EJ Houtgast, V Sima, G Marchiori, KLM Bertels, and Z Al-Ars. Power-Efficient Accelerated Genomic Short Read Mapping on Heterogeneous Computing Platforms. In *Proc. 24th IEEE International Symposium on Field-Programmable Custom Computing Machines*, Washington DC, USA, May 2016.

[7] Heng Li. Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.

[8] Lukasz Ligowski and Witold Rudnicki. An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[9] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, and R. Li. SOAP3: Ultra-Fast GPU-Based Parallel Alignment Tool for Short Reads. *Bioinformatics*, 28(6):878–879, 2012.

[10] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[11] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. CUDASW++ 3.0: Accelerating Smith-Waterman Protein Database Search by Coupling CPU and GPU SIMD Instructions. *BMC bioinformatics*, 14(1):117, 2013.

[12] TF Smith and MS Waterman. Identification of Common Molecular Subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[13] ZD Stephens, SY Lee, F Faghri, RH Campbell, C Zhai, MJ Efron, R Iyer, MC Schatz, S Sinha, and GE Robinson. Big Data: Astronomical or Genomical? *PLoS Biology*, 13(7), 2015.