A Novel High-Throughput Acceleration Engine for Read Alignment

Yu-Ting Chen,* Jason Cong,* Jie Lei*† and Peng Wei*
*Computer Science Department, University of California, Los Angeles, USA
{ytchen, cong, jielei, peng.wei.prc}@cs.ucla.edu
†State Key Laboratory of Integrated Services Networks, Xidian University, China
jielei@mail.xidian.edu.cn

***Abstract*—The Smith-Waterman (S-W) algorithm is widely adopted by the state-of-the-art DNA sequence aligners. Existing wavefront-based methods ignored the fact that the S-W algorithm is fed with significantly varied-size inputs in modern aligners, in which the S-W algorithm is further optimized by exerting extensive pruning. In this paper we propose an architecture, tailored for varied input sizes as well as harnessing software pruning strategies, to accelerate S-W. Our implementation demonstrates a 26.4x speedup over a 24-thread Intel Haswell Xeon server, and outperforms wavefront-based implementations by up to 6x with the same FPGA resource.**

***Keywords*-read alignment; Smith-Waterman; FPGA; HLS; multilevel scheduling**

I. Introduction

Next-generation sequencing (NGS) technologies have attracted a large amount of attention from both researchers and clinicians. Human DNA samples are chopped into billions of small fragments, called reads, and a sequencer determines the order of each read's nucleotides. A software aligner then maps the billions of sequenced reads onto a reference human genome, which entails tremendous computational challenges. A read typically consists of hundreds of nucleotides or base pairs (bps), while the reference genome contains 3.2 billion bps [1].

State-of-the-art read aligners, such as BWA-MEM [2] and Bowtie 2 [3], carry out the read mapping in two steps. First, each read is fragmented into small pieces called seeds, and mapped to the reference genome. The mapping of seeds to the reference genome must be exact, i.e., no gap or mismatch is allowed. A backward search algorithm based on the Burrows-Wheeler Transform (BWT) [4] takes only O(m) time for a seed of length m to map to the super-long reference genome, independent of the size of the reference genome, and therefore is employed by almost all contemporary sequence aligners.

In the second step, each seed map gets extended leftward and rightward to span the entire read. The extensions allow inexact mappings, in which gaps and mismatches are allowed. A predefined scoring function is provided for evaluating the effectiveness of inexact mappings. Only the ones that achieve high enough scores are recorded in the output. The Smith-Waterman (S-W) algorithm, a dynamic programming (DP) algorithm with quadratic time complexity, is a commonly used approach for the inexact mapping step. It is the main computation bottleneck in state-of-the-art aligners like BWA-MEM (30%-50% computation time) [2]. In this work we focus on accelerating the S-W algorithm in BWA-MEM. The methodology, however, can be applied to other aligners, such as Bowtie 2 [3] and LAST [5].

The S-W algorithm is inherently anti-diagonal parallelizable, since the elements along the anti-diagonal direction in the 2D table used for the DP procedure are independent of each other [6]. This feature, called wavefront, has been explored in different ways on different platforms [7][8][9][10]. It works fairly well when input sizes are homogeneous. However, the wavefront-based techniques applied to the general S-W algorithm do not fit well for the optimized version used in BWA-MEM. First, a huge number of reads at the billion scale need to be processed in a high-throughput fashion. Conventional approaches overemphasize the inner-task parallelism while neglecting the task-level parallelism. Second, conventional wavefront-based architectures cannot utilize computational resources efficiently when the input sizes vary sharply. Third, the pruning heuristic used in BWA-MEM prevents the wavefront-based techniques from being adopted directly and efficiently.

In this paper we propose an architecture to address these issues. The novelties of our architecture can be summarized as follows: (1) we propose an array-based architecture for processing the enormous number of reads in a high-throughput fashion, adapting better to inputs with widely varied sizes; (2) we provide a two-level hierarchical architecture for resource management, reducing the amount of resources needed for synthesizing bus interfaces while satisfying the off-chip bandwidth demand; (3) our design supports the pruning technique, shortening the runtime of the S-W algorithm significantly.

Our FPGA implementation demonstrates a 26.4x speedup compared to a 24-thread Intel Haswell Xeon server for the S-W algorithm in BWA-MEM. We also show that our design outperforms wavefront-based S-W implementations by up to 6x with the same FPGA resource utilization.

II. Basis of Our Approach

Our acceleration engine design is based on the following three important observations derived from an analysis of the

S-W implementation in BWA-MEM.

Observation 1: Enormous Task-Level Parallelism

A sequencer can generate billions of reads from a single individual for analysis in today's NGS flow. The huge amount of data generates enormous task-level parallelism, prompting us to reexamine the conventional wavefront technique for S-W acceleration. The conventional wavefront technique exploits the inner-task anti-diagonal parallelism to maximize the speedup for a single task. However, the wavefront implementation is not optimal when the task-level parallelism is several orders of magnitude larger than the inner-task parallelism. For an accelerator platform with limited resources, e.g., a FPGA board with a certain amount of LUTs (DSPs, BRAMs, etc.), we must decide if the resources should be allocated for exploiting the task-level parallelism or the inner-task parallelism.

Observation 2: Significantly Varied-Size Inputs

For simplicity without losing generality, we abstract the total resources of an accelerator platform into a given number of unified processing elements (PEs). We also assume that each PE is capable of producing one value per cycle to fill the 2D table in the DP algorithm. A *kernel* is composed of a group of PEs and can be assigned to execute one S-W task. We denote mxn as a pair of input strings for the S-W algorithm with length m and n.

The sharply varied input sizes of S-W in BWA-MEM result in a considerable waste of resources in wavefront-based designs. For example, a kernel of 10 PEs is only able to reach a maximum of 65% resource utilization for a 13x103 input because the length of the maximal anti-diagonals (13) is not divisible by the number of PEs (10). It takes 2 cycles for 10 PEs to fill an anti-diagonal with 13 elements. Figure 1 provides a histogram of the sizes of the shorter strings (bounding the maximum degrees of parallelism) over 10M inputs of randomly selected BWA-MEM S-W tasks. The sizes range from one to 84, and none of them has more than 5% of the 10M inputs. The significant diversity of input sizes makes it prohibitive to choose one or a few kinds of PEs to avoid wasting resources. A better choice is to have each kernel restricted to only one PE, which means the anti-diagonal parallelism gets totally ignored.

Observation 3: Pruning Strategies

Derived from the X-dropoff pruning strategy in BLAST [11], BWA-MEM's pruning strategy is able to save over 50% in computation efforts for S-W tasks, as illustrated in Figure 2. However, the pruning strategy destroys the basis of the anti-diagonal parallelism, as described in detail in [2]. Moreover, the results generated by the optimized S-W algorithm in BWA-MEM are slightly different from those obtained by the standard S-W algorithm. This increases the difficulty of integrating existing wavefront-based work into BWA-MEM for the concern of the result credibility. Even if the difficulty of integration can be overcome, the potential speedup from pruning would have to be sacrificed due to its incompatibility with the wavefront technique. It will be even worse when the sizes of seeds become longer, which is the future trend for NGS.

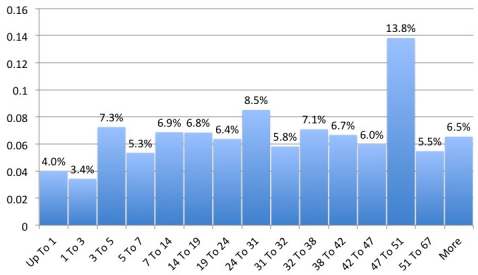

Figure 1. Histogram of the lengths of the shorter input strings collected from 10 million randomly selected BWA-MEM S-W tasks.

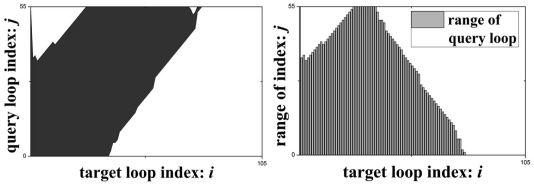

Figure 2. A 55x105 BWA-MEM Smith-Waterman task. The general S-W algorithm requires filling up a 55x105 matrix (5775 elements), but only 2836 elements (49%) were actually filled with the help of pruning. The black area in the left graph illustrates the elements that got filled, and the right graph shows how many elements for each target loop index are actually calculated.

III. ARCHITECTURE DESIGN AND IMPLEMENTATION

A. Overall Architecture

Our accelerator engine consists of multiple PE arrays. Each PE array has a task distributor connected to the off-chip memory via an AXI bus interface, as shown in Figure 3. Each PE acts as one kernel and can take one S-W task at a time to maximize throughput.

Before the accelerator starts to operate, the host processor assembles a set of query and target sequence pairs and streams them to the on-board DDR3 memory via the PCIe bus. After that, each PE array's task distributor fetches a certain number of sequence pairs and distributes them to the idled PEs. The mapping results are stored in the on-board memory and then sent back to the host.

B. Processing Element (PE) Design

By using the high-level synthesis methodology, the hardware structure of a PE is created based on the S-W software code obtained from the software implementation of BWA-MEM. Unlike wavefront-based solutions that look completely different from the software structure, our design naturally follows the original software structure, which considerably shortens the development cycle.

To maximize throughput, we make the initiation interval (II) of the PE design be equal to one (II=1), i.e., calculating one cell of the score matrix per cycle. Moreover, a conditional branch logic is implemented to realize pruning. It enlarges the size of each PE by 20%, but reduces computation effort by over 50%.

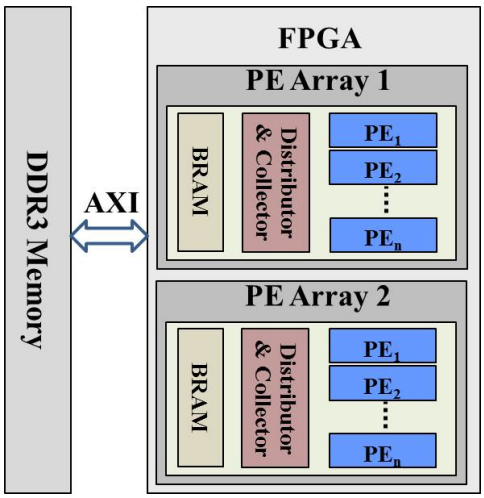

Figure 3. An overall architecture of the proposed accelerator engine.

C. Two-Level Task Scheduling

The first level of task scheduling resides in a PE array. A PE array includes three types of major components: (1) a *task distributor*, (2) a set of *PEs*, and (3) a *result collector* (as shown in Figure 4). The *task distributor* fetches data from the on-board DRAM to the on-chip BRAM and dynamically dispatches the tasks to each PE through a FIFO. The *result collector* receives mapping results from each PE through a FIFO and packs them together for the host.

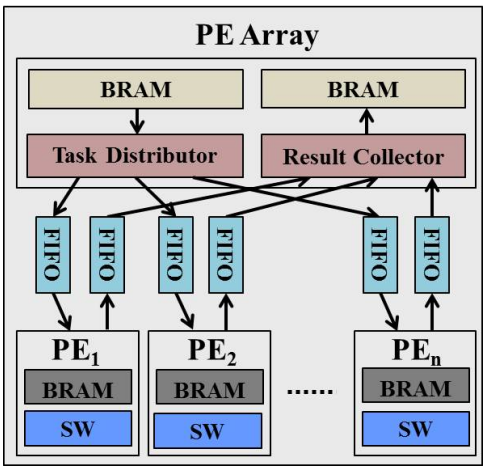

Figure 4. The design of a PE array.

If a centralized *task distributor* is not provided, each PE needs to fetch data from on-board memory independently. This incurs a significant (25%) area overhead per PE to synthesize its own AXI interface. To reduce the overhead, we use only one AXI bus interface for each *task distributor* per PE array. A PE array, acting as an AXI bus master, fetches a group of data that satisfy the off-chip bandwidth demand of all PEs in the array. We also implement ping-pong buffering by using a pair of BRAM blocks for better performance. After the data are prefetched to BRAM, the distributor checks the flag of the FIFO of each PE one by one to find an available PE. The *task distributor* then transfers the S-W tasks from BRAM to the FIFOs of the PE. This process continues until all the tasks are processed. The *result collector* continues monitoring the output FIFO of each PE and obtaining results until all the results are received.

If the number of PEs is small, the one-level task scheduling is sufficient. However, when the number of PEs is increased to a certain point, e.g., 50, the distributor becomes a performance bottleneck due to its round-robin task scheduling scheme. Therefore, we introduce a two-level task scheduling scheme which feeds tasks to multiple arrays, each with its own *task distributor*. This two-level hierarchy provides us with a scalable design methodology for obtaining scalable speedup.

IV. Experimental Results

A. Experimental Setup

Our design is described in HLS C and synthesized using the Vivado HLS [12]. The accelerator engine contains two PE arrays, each composed of 50 PEs, and can run at 150MHz on the Xilinx VC707 FPGA evaluation board. The FPGA resource utilization is approximately 65% (LUTs: 65.4%, FFs: 30.7%, BRAMs: 21.7%). We use an input with 100M reads, which is a subset collected from a real individual human genome sample (about 500M reads, 120GB FASTQ file) with breast cancer (HCC1954). We verify the outputs of our FPGA implementation with the seed extension step of BWA-MEM to ensure correctness.

B. Speedup Over Software-Based BWA-MEM

To demonstrate the speedup of our accelerator engine over the original BWA-MEM software, an Intel Haswell Xeon server with two 6-core CPUs is used. The server can run 24 threads in parallel, with hyper-threading support. We compare the execution time of our FPGA implementation to pure software-optimized S-W algorithm runtimes with 1, 2, 4, 8, 16 and 24 threads. To make a fair comparison, only the computation time of the S-W calls (after loading inputs from memory and before storing outputs to memory) is collected.

Figure 5 shows the performance comparison between our FPGA design and the software BWA-MEM S-W algorithm with results normalized to the single-threaded CPU performance. Our FPGA design outperforms the 24-thread CPU by 26.4x.

C. Speedup Over Wavefront-Based Designs

To highlight the advantage of our design over wavefront-based techniques, we compare the runtime of our accelerator engine to those of a set of wavefront-based FPGA implementations described in [9]. To arrive at a fair comparison, we implement all designs with roughly the same FPGA resource utilization.

Table I shows the wavefront-based designs that we implemented for comparison. A design containing n kernels with m PE per kernel is denoted by mPxnK. We use the 32Px5K configuration as the base unit for comparison. We find that the performance of the wavefront-based designs generally decreases when the number of PEs per kernel (m) increases, though the performance does not decrease fully monotonically, as shown in Figure 6. In general, this verifies that the wavefront-based designs have worse PE utilization rates compared to our design.

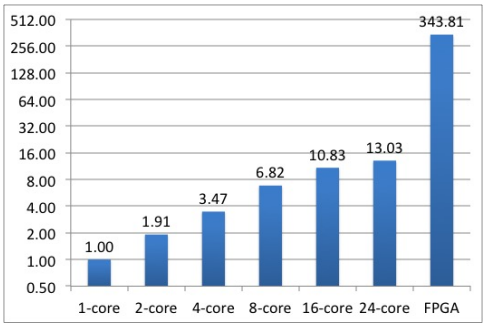

Figure 5. Performance comparison between our FPGA design and the multithreaded BWA-MEM software. A base-2 logarithmic scale is used for the Y axis to denote the normalized performance.

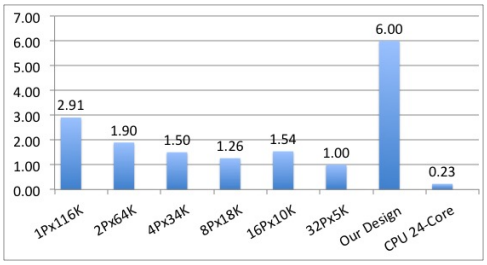

Figure 6. Performance comparison between our design and a set of wavefront-based designs. The Y axis denotes the performance normalized to that of the 32Px5K configuration.

Furthermore, Figure 6 also demonstrates the power of pruning. When the wavefront design goes to the extreme where a kernel is restricted to only one PE (1Px116K), the only difference between our design and the 1Px116K design is whether the pruning logics are realized. Compared to the 1Px116K design, our design shows about 2x speedup. Our current design uses reads with a length of 101bp. Since the NGS flow will generate longer reads in the future, the performance improvement from pruning is expected to be much more significant.

V. Conclusions

In this paper we design an architecture to efficiently accelerate the S-W algorithm, which is the computation bottleneck in the state-of-the-art read aligner, BWA-MEM. Massive task-level parallelism, sharply varied input sizes, and software-pruning strategies are all well supported in our design. Our FPGA implementation demonstrates a 26.4x speedup over a 24-thread Intel Xeon server, as well as up to a 6x improvement over wavefront-based implementations.

Table I
Configuration of Different Designs

Design	Resource Utilization (%)	#Kernels	#PEs/Kernel
1Px116K	64.9	116	1
2Px64K	64.6	64	2
4Px34K	64.9	34	4
8Px18K	66.8	18	8
16Px10K	73.1	10	16
32Px5K	72.6	5	32
Our Design	65.4	100	1

Acknowledgment

This work is partially supported by the Intel Corporation with the matching fund from the NSF under the Innovation Transition (InTrans) Program (CCF-1436827), and by the National Science Foundation of China (No. 61301287).

References

[1] E. R. Mardis, "The impact of next-generation sequencing technology on genetics," *Trends in Genetics*, vol. 24, no. 3, pp. 133 – 141, 2008.

[2] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *ArXiv e-prints*, Mar. 2013.

[3] B. Langmead and S. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Method*, vol. 9, pp. 357–359, 2012.

[4] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Tech. Rep., 1994.

[5] M. Frith, M. Hamada, and P. Horton, "Parameters for accurate genome alignment," *BMC Bioinformatics*, vol. 11, no. 1, p. 80, 2010.

[6] E. Edmiston, N. Core, J. Saltz, and R. Smith, "Parallel processing of biological sequence comparison algorithms," *IJPP*, vol. 17, no. 3, pp. 259–275, 1988.

[7] T. Preusser, O. Knodel, and R. Spallek, "Short-read mapping by a systolic custom FPGA computation," in *FCCM*, 2012, pp. 169–176.

[8] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms," in *ISCA*, June 2014, pp. 517–528.

[9] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform," in *HPRCTA*, 2007, pp. 39–48.

[10] B. C. Lam, C. Pascoe, S. Schaecher, H. Lam, and A. D. George, "BSW: FPGA-accelerated BLAST-wrapped Smith-Waterman aligner," in *ReConFig*, 2013, pp. 1–7.

[11] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403 – 410, 1990.

[12] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *TCAD*, vol. 30, no. 4, pp. 473–491, 2011.