An MPSoC Based Embedded System Solution for Short Read Genome Alignment

Vikkitharan Gnanasambandapillai Arash Bayat Sri Parameswaran

University of New South Wales, Australia
{vikki.g a.bayat, sri.parameswaran}@unsw.edu.au

Technical Report
UNSW-CSE-TR-201710
November 2017

UNSW
SYDNEY

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Computational needs for genome processing are often satiated by enormous servers or by cloud computers. Even though there has been some work in implementing some aspects of genome processing in GPUs and FPGAs, they are often accelerators for such servers and not stand-alone systems. In this paper, for the first time, we present a method to entirely move the alignment process to embedded processors. Such a system is useful in a variety of situations where significant networked infrastructure is not available, and where privacy is a concern. A ring pipelined processor architecture for short read alignment, based on partitioned genome references is shown. A timeout based alignment method is proposed to prevent unnecessary exhaustive search. The proposed partitioning method allows an entire human genome to be processed using small embedded processors. Experimental results show that the proposed solution speeds up the performance by approximately seven times with 16 embedded processors when compared to a linear pipelined system. It is expected that the proposed solution will to lead to a portable genome analyzing device, significantly reducing cost and testing time.

1 Introduction

Genome analysis has two broad steps: DNA sequencing and the computations necessary for the genome processing. Latest technology developments in DNA sequencing reduce size and price of sequencing machines and speeds up the sequencing [17]. Furthermore, the performance gap between DNA sequencing and genome processing computations is increasing [7]. Also, thumb size commercial sequencers are now becoming available (for example, MinION sequencer [18]). However, current genome processing computational platforms are still large and expensive. We are still reliant on cloud servers or large computational servers to process the genome. If a genome processing computing platform is designed to be smaller and faster, portable genome processing devices will be possible.

Developing low-cost, faster, portable genome computing devices bring cost, time and availability advantages to genome tests. Reducing genome sequencing cost will allow greater prevalence of genome testing, which allows for personalized treatment based on an individual's genome. Reducing genome testing time will allow a patient's genome condition to be monitored frequently [11]. Portability will allow for field monitoring allowing identification of the root cause of the fast spreading epidemics from affected areas, particularly from remote locations [19]. With existing genome computing platforms, these benefits are limited since considerable network infrastructure as well as costly computing platforms are necessary. The non-reliance of cloud based systems further addresses privacy concerns. While FPGA based systems are fairly common to accelerate parts of the genome computations, we find that with ever improving algorithms, the redesign cost of repurposing FPGAs to be deeply prohibitive.

To create portable systems, genome computation must be moved to smaller embedded processors. However, genome computation on embedded processors is challenging for the following reasons: (1) embedded processors are typically less powerful, they have smaller bit widths and a smaller memory footprint; and (2), in embedded processors, existing genome algorithms are slow and not scalable as these algorithms are not intended for embedded systems. For example, it is not possible for the BWA-aln algorithm (a popular short read genome aligner) to be executed on a single Tensilica processor (since processor's memory footprint is too small). Smaller caches in embedded processors further worsen the situation.

However, with the advent of MPSoCs with algorithmic and architectural modifications, embedded processor based solutions could meet the need for performance, cost, and portability. Pipelined processor architectures are more suitable for genome processing due to its high performance, lower cost, smaller area and lower power consumption. Pipelined processor architectures are widely used for many computationally intensive applications such as video and image processing [8, 10, 9].

We present a ring pipelined processor architecture for short read genome alignment using embedded processors. We partition the human genome reference to reduce memory accesses and the required memory footprint per processor. Our solution speeds up the short read alignment and results are shown for one million simulated reads aligned to the full human genome. Such an architecture enables the full aligning of short reads to the full genome.

1.1 Contributions

The contributions of this paper are as follows:

- for the first time, we show that genome processing can be performed on embedded processors;
- to enable the implementation on embedded processors, a novel ring pipelined architecture for short read genome alignment algorithms is presented;
- a method for load balancing amongst processors is given;
- a method (partitioning the reference genome) to reduce memory accesses and footprint per processor is presented; and,
- the concept of a timeout threshold is introduced to enable faster alignment.

The rest of the paper is organized as follows. Section 2 summarizes the background of short read genome alignment. In Section 3, short read genome alignment performance improvements are analyzed. The proposed methodology and architecture are explained in Section 4 and 5. In Section 6, MPSoC embedded processor implementation is described. Experimental setup and results are explained in Section 7. Finally, Section 8 concludes this paper.

2 Background

The entire genome of a living being is necessary for genealogy and health analyses [20]. Since assembly of the genome from scratch (using the reads) is a time-consuming process, an existing genome (referred to as the reference genome) is used for reassembly of other individuals of the same species. The short reads are aligned by comparing the reads and the reference [13]. The human genome is around 3.2 billion base pairs and number of reads to be aligned is more than a billion [4] (to speed up experiments, we use one million reads).

To speed up the alignment problem, graphs theory and dynamic programming are used to index the reference and/or reads. Read alignment is one of the most time-consuming algorithms in genome computations, therefore, researchers continuously examine ways to reduce the execution time of alignment algorithms [4].

One way of reducing the execution time is indexing the reference to facilitate faster searching of the reference genome for a given read. There are two types of indexing: hash-index and Farregina-Manzini (FM) Index [4]. In hash-index, a hash table stores the location of each fixed length subsequence taken from the reference genome. Hash index size of the human genome is more than 50 GB [21]. Modern read aligners use FM-index with Burrows-Wheeler Transform as FM-index requires smaller memory (less than 4GB) [12, 13] and is faster when compared to the hash-index. Please see [4] for more details about short reads genome alignment.

The Burrows-Wheeler Aligner (BWA) is one of the fast and accurate alignment software suites available today and is widely used for read alignment [5]. "BWA-aln" is a short reads genome aligner based on FM-index. It requires only 2.3 GB memory for the index and associated auxiliary data for the full human genome. "BWA-aln" is MESGA's software component.

"BWA-aln" has three steps. In the first step, the FM-index and auxiliary data are generated for the reference genome. This is a once only process and can be done off-line as the reference genome is fixed. In the second step, the best

alignment position is identified, this is the main and more time-consuming step so we examine methods to speed up this step. Finally, alignment coordinates are reported according to a standard format.

Hereafter, short reads are referred as reads and genome alignment is referred as alignment.

3 Related Work

As computer clock speed has plateaued, alternate performance improvement solutions have been developed for speeding up read alignments. Such performance improvements have included multi-core CPUs and many-core GPUs [16]. Other techniques to speed up alignment have included software parallelism and hardware accelerators.

3.1 Performance Improvement by Parallelism

Multi-threaded read alignment algorithms make use of available processors in a system to gain performance [15]. Modern aligners are parallelized using multi-threading [4]. However, performance does not increase with the number of processors due to intensive random memory accesses and unbalanced loads between threads [2]. To reduce the memory accesses some researchers duplicate the reference genome for high-performance computers [5] and some researchers partitioned the reference to reduce the memory footprint [2].

For example, Chen et al. [5] implemented reference duplication for a hybrid system (consisting of a 12 core Xeon processors and a 60 core Xeon Phi co-processors). Few cores are grouped together as a logical unit and share a single reference. Input data (short reads) are split into several subsets according to the number of logical units and a subset is processed in a logical unit. Individual alignment time for a Xeon processor and Xeon Phi processor are used to determine the size of the subsets of balancing the loads in the hybrid system.

GNUMAP [2] partition the hash index to reduce memory footprint in multi-node high-performance clusters. However, performance is compromised by as much as 10 times.

3.2 Short read Alignment Hardware Accelerators

Hardware accelerators are considered as another solution to increase the performance of read alignment, for example, CUSHAW, CUSHAW2-GPU and FHAST [3]. Graphical Processing Units (GPUs) and Field Programming Gate Arrays (FPGAs) are utilized as hardware accelerators [4, 22].

GPUs together with CPUs are widely used in many parallelized short read aligners. For example, CUSHAW is a short read algorithm which is based on the FM-index [16]. Compared to other CPU based aligners, CUSHAW has speed advantages for less than 51 base pair reads but it is slow for longer reads. CUSHAW does not support gap alignment. However, its newer implementation, CUSHAW2-GPU, supports limited gap alignments [15]. CUSHAW2-GPU makes use of heterogeneous computing architecture of CPUs and GPUs. Read alignment algorithms are parallelized as inter-track hybrid CPU-GPU mode.

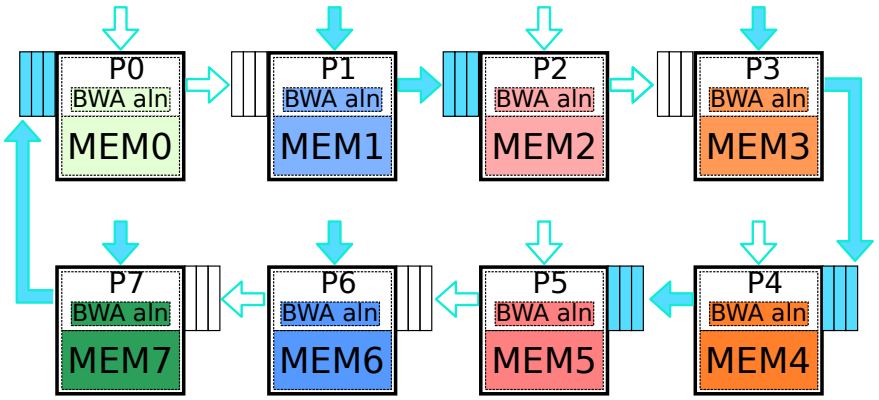


Figure 3.1: MESGA's Ring Pipelined Processor Architecture.

Also, GPUs, CPUs and system memory hierarchy are utilized for the optimization.

Many researchers use FPGAs as a hardware accelerator in high-performance computers [3, 6]. The speed up is achieved by executing parallel streams in FPGAs. One such example is FHAST [6], consisting of dual-core Intel Xeon CPU and four Virtex-6 FPGAs. The algorithm is based on Bowtie, which is another popular FM-index based aligner [12]. Speed up of 12× is achieved compared to native Bowtie executing on eight cores. Internal queues are used to buffer the data and thus reduce memory latency. To support n number of mismatches, $n + 1$ exact matching blocks are used. Although it supports up to 101 base pair reads, it covers only 3% of the human genome.

Compared to the above works, MESGA is better in terms of implementation, purpose, and coverage. MESGA is based on MPSoC, which is cheap, lower power and adaptable compared to GPUs and FPGAs. With customized architecture for genome algorithms, MPSoC outperforms GPUs and FPGAs.

Duplicating the reference genome as in [5] is suitable for the high-performance system but not for embedded systems as it is a memory expensive method. As far as we know, the reference genome has not been divided in the FM-index based aligners. In MESGA the FM-index is divided to improve the parallelism and run read aligner for the full human genome on embedded processors.

GPU and FPGA accelerators have drawbacks. Limited memory in these accelerators do not always allow the full human genome alignment [3]. Code modification too is a challenging tasks. Software algorithms need to be extensively rewritten to support GPUs and algorithm need to be fine-tuned according to the GPU architecture. Similarly, for FPGA accelerators, algorithms need to be converted to hardware description languages. When algorithms change (as they tend to), extensive code modifications are necessary [3]. Our solution, MESGA, is suitable for any software algorithms with the minimal code changes. In other systems, design parameters such as number of mismatches and gaps are limited and cannot be changed. But MESGA supports all the features of the original software.

In [5], processors' loads are balanced using static timing and balancing is not perfect for real data. In MESGA, loads are balanced amongst the processors by ring pipelined architecture.

4 Methodology

MESGA utilizes four methods to reduce the short read alignment time. These are (1) MPSoC based ring pipelined processor architecture; (2) partitioning the reference genome, indexing the partitioned genome, and storing the smaller index; (3) a timeout based alignment failure detection; and (4), cache optimization. The partitioning the reference genome enables short read alignment for the full human genome using embedded processors with limited memory footprint. The timeout avoids excessive searching caused by the genome partition (see Section 6 for the details).

In MESGA, memory footprint and accesses are reduced by carefully partitioning the reference genome (if four processors are used the genome is divided into four parts- with some overlap between the partitioned genomes). Then each of the partitioned genomes are then indexed using the BWA index [13]. This index is usually one-fourth the size (for the division of four) and can be stored within a single processor's memory footprint.

Each processor accesses one of the partitioned indices, so the memory accesses within each processor's local memory are significantly reduced compared to the original "BWA-aln" (approximately two times less if four processors are used, since unmatched reads take a little longer searching for possible matches). In MESGA's pipelined architecture, processors are identical and execute the same algorithm ("BWA-aln") with different partitioned reference. Each processor tries to align reads to its own partitioned reference. Some reads are transfers to the next processor based on a criterion. Note that, the passing criterion (calculated using BWA-aln's scoring system, which is automatically generated) has two quality thresholds, 105 & 85 (perfect alignment score is 125). A read is not passed to the next processor, if the alignment score exceeds 105. A read is passed, but the current local alignment is kept if the score is between 85 and 105. Finally, if the alignment score is below 85, then the read is passed without the local alignment being recorded. If a read has been processed by all processors and not once exceeded 105, then the best available alignment is used.

5 MESGA Architecture

MESGA's ring pipelined architecture is shown in Fig. 3.1. In this figure, preprocessors ids are indicated as Px, local memories are shown by MEMx, queues are represented by array of rectangles (white is empty queue and filled in blue color is a non-empty queue) and arrows represent data transfer paths (white is no data transfer and filed in blue color is data transfer is active). Eight Processors are connected in a ring and each processor has an input queue (which is the output queue for the front processor), an external interface and a local memory. Processors communicate unidirectionally using the queues. The external interface provides a connection to the storage device where all the reads are stored. Each processor can either receive from the storage device or the input queue. The storage device is only read if the input queue is empty. The storage device access time is excluded in our experiments.

MESGA's novel architecture provides higher processor utilization and better load balancing than a linear pipeline. In a linear pipelined architecture, a processor only reads from its input queue (except for one processor which reads

Table 5.1: MESGA Processor Configuration Range.

	Min Configuration	Max Configuration
Speed (MHz)	200	1092
Processor Size (mm^2)	0.05	0.08
Processor Size(gates)	50168	85450
Processor Power (mW)	4.7	40

the storage device) and a processor is idle when its input queue is empty. However, in MESGA a processor receives data from the external interface when the input queue is empty thus reducing the chances of a processor being idle. For example, in Fig. 1, P1 is active although its input queue is empty as data come from the external interface.

6 MESGA Implementation

6.1 MESGA Hardware Implementation

MESGA hardware is simulated using the Xtensa Modeling Protocol (XTMP) simulator [1]. For time-sensitive experiments such as cache configuration analysis, clock-accurate simulation is used and for other experiments, XTMP "Turbo" feature was used to reduce the simulation time.

MESGA consists of up to sixteen 32-bit Tensilica LX6.0 processors. Processors are customized using the Xtensa development tool (RF-2016). By default, each processor has 2GB RAM, 16MB ROM, 32kB instruction cache and 8kB data cache with associativity 2. Inter-processor interface is a 32 bit width queue of 64 kB (64 kB were chosen so that there was no blocking, a smaller size may be chosen in the future after more analysis). Partitioned reference index and auxiliary data are fetched from each processor's local RAM. RAM is simulated with write and read delay of 30 clocks and 10 clocks for a read request and block read request. The reads are stored in multiple files; the number of files equals the number of processors. These files are connected to each processor via the external interface.

Table 5.1 summarizes the processor frequency, size, and power range. In all our experiments, the clock frequency is set to 982 MHz and with this frequency, the peak power is 37.5mW. The area of the processor is 0.08 mm^2 at 28nm technology.

6.2 MESGA Software Implementation

MESGA's executes the latest "BWA-aln" (0.7.15). Some code modifications and additional data structures are added to support embedded processors, pipelined architecture, and partial reference. Alignment quality based transferring reads and timeout based alignment failure detection are the other code changes.

In Algorithms1 & 2, qi and qo are input and output queues. Algorithm 1 shows the overall input data handling and task completion of the system. Lines 1 & 7, prioritizes the input queue to the external interface and reduces the probability of the data stall due to previous processor's output queue being full. Lines 3 & 9 makes sure none of the reads are processed by a processor twice. Line 14 sets a bit in a flag to indicate the processor has processed all reads from

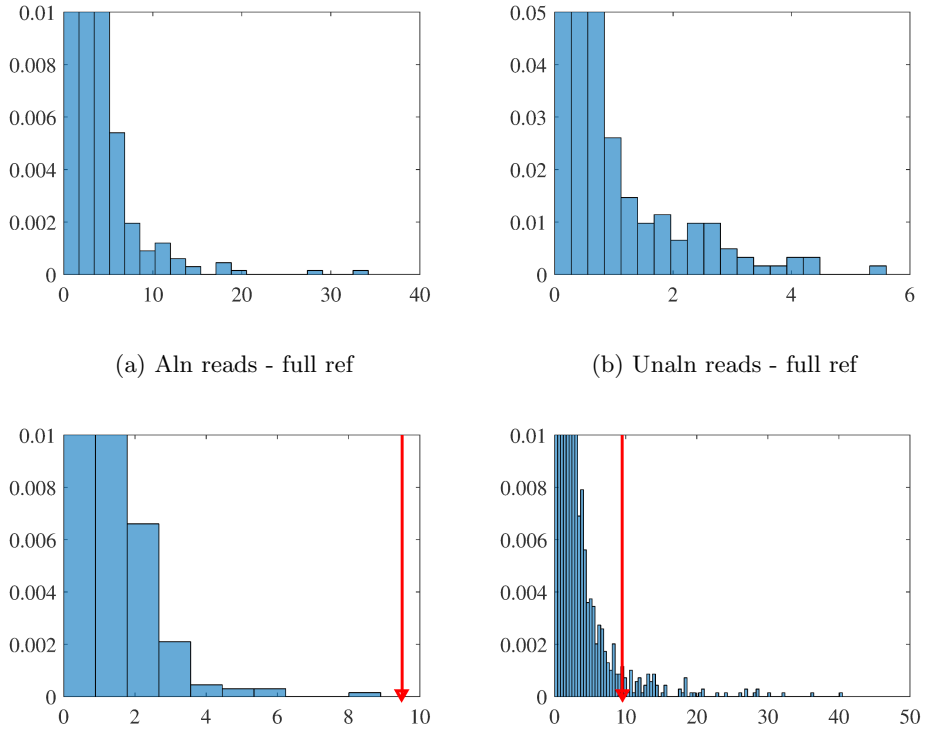
Algorithm 1 ReceiveNewData

```
1: if qi data available then
2:    fetch data
3:    if the data has processed by this processor then
4:       ReceiveNewData
5:    end if
6:    AlignRead
7: else if file data available then
8:    Read data
9:    if the data has processed by this processor then
10:      ReceiveNewData
11:   end if
12:   AlignRead
13: else
14:   Set this processor finish flag
15:   if all other processors finish then
16:      Finish alignment
17:   else
18:      ReceiveNewData
19:   end if
20: end if
```

its file, and if all bits corresponding to each processor in this flag are set then all inputs from files have been processed. Based on this flag and all queues' empty status, processors complete the "BWA-aln" algorithm. Algorithm 2 shows time out based failure detection and transferring a read to the next processor based scoring.

Algorithm 2 AlignRead

```
1: if time < time out threshold then
2:    if score < lower threshold then
3:       push data to qo
4:    else if score < higher threshold then
5:       record alignment result
6:       push data to qo
7:    else
8:       record the alignment result
9:    end if
10: else
11:   alignment fails
12: end if
```



(a) Aln reads - full ref

(b) Unaln reads - full ref

(c) Aln reads - partial ref

(d) Unaln reads - partial ref

Figure 6.1: Read alignment execution time distribution.

Aligning reads within a partial reference may increase execution time. Most reads will not belong to the reference in the processor, so most reads are unaligned. "BWA-aln" takes longer time for a read with many mismatches and gaps. Thus, it takes a long time to decide that a read cannot be aligned. Fig. 6.1 shows execution time distribution for four cases: (a) is for successfully aligned reads with the full reference; (b) for unaligned reads with the full reference; (c) is for successfully aligned reads with partial reference (divided into 16 sections); and (d), for unaligned reads with partial reference. The x-axis is number of execution clocks in million for individual reads and the y-axis is the probability of occurrence. Note that the x and the y axes have different range to show the histogram details clearly. Execution time in (c) & (d) of aligned reads reduces and increases respectively when the reference is divided. The threshold is set as the maximum execution time of aligned reads (shown by the vertical lines) with partitioned reference.

The timeout threshold varies with the number of divisions and is shown in Fig. 6.2. In Fig. 6.2 the reference is divided into 2, 4, 8 or 16, which is indicated on the x-axis, and the y-axis represents timeout threshold in million clocks. This is a decaying function.

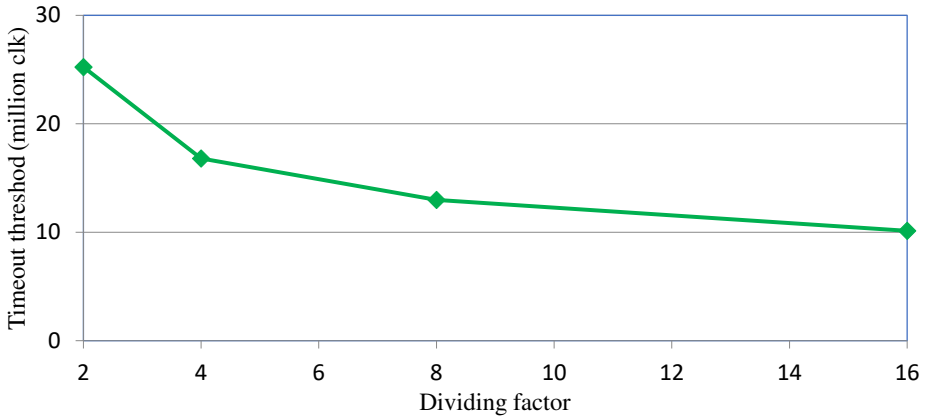


Figure 6.2: Timeout variation with number of reference division.

7 Experimental Setup and Results

To evaluate the proposed method's performance improvement and alignment accuracy, one million (all are 125 base pair length) synthetic reads were aligned to the full human genome. The reads were simulated from the human genome using dwgsim [14] (similar to the the original "BWA-aln" benchmark tests [13] (0.09% SNP mutation rate, 0.01% indel mutation rate, and 2% uniform sequencing error rate).

Reads were divided equally into N groups and saved into separate files. N is equal to the number of processors in the pipeline (N = 2, 4, 8 or 16). The full human genome reference was partitioned into N with overlapping regions of 250 base pairs. Each segment was pre-processed and indexed separately off-line.

7.1 Execution Time

To compare the performance, four configurations were used with 2, 4, 8 or 16 processors. First, embedded processors are in a linear pipeline and each processor has equal size of partitioned reference. All reads were fed to the first processor. All reads are passed to next processor except the reads which were aligned exactly (this is to simulate the original condition of running "BWA-aln" with a full genome). Second, optimized linear pipeline, reference was divided in progressively increasing size to balance the load between front end processors and back end processors. Reads are passed to next processor according as described in Section 4. The third experiment is with MESGA without timeout and the final one is MEGSA with timeout.

Fig. 7.1 shows the speed up comparison for the four experiments. Single processor timing was estimated, as the processor memory is not enough to hold the full human genome. The single processor takes 16.8 hours to align 1 million 125 base pair reads. Meantime, with 16 processors, MESGA speeds up seven times with timeout and five times without timeout. The optimized linear pipeline improves the performance by three-folds and linear pipeline with equal reference shows almost no improvement for 16 processors (this is due to idling of many of the processors).

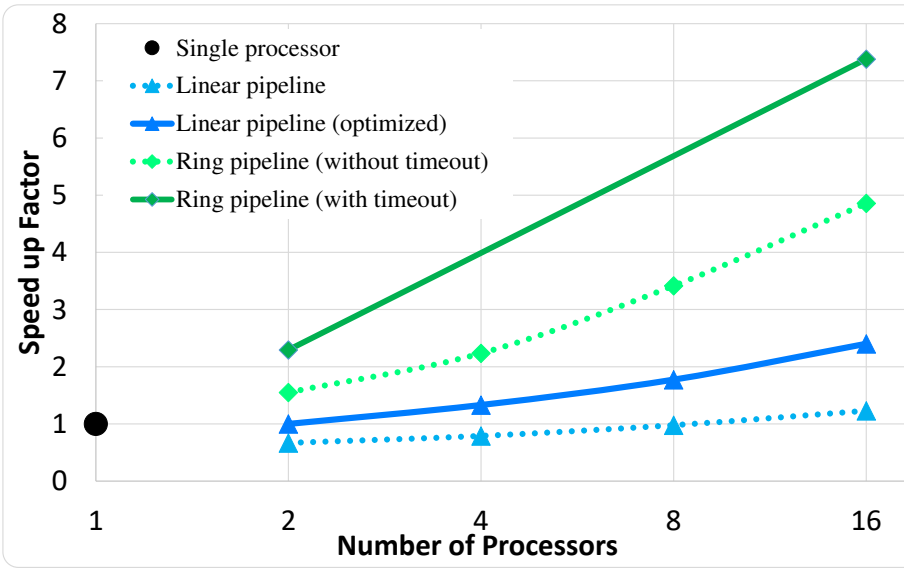


Figure 7.1: Read alignment speed up comparison.

7.2 Accuracy

To verify the accuracy of the output after code modification, the individual alignment scores generated by baseline test (original "BWA-aln" running on an Intel server) and MESGA (with timeout) are compared. Individual MESGA read alignment score is compared with the baseline test. All alignment scores of MESGA were the same as the baseline test score.

7.3 Cache Configuration and Power Analyses

Effect of the cache configuration on the performance of MESGA was analyzed in XTMP simulation platform with cycle accurate timing. As the simulation takes a longer time, this experiment was conducted with only 10,000 reads. Cache size was changed from 0 to 32 kB and 1,2 and 4 cache associativities were tested. Table 7.1 summarizes the execution time for different data cache configurations. The top row shows the cache size and the left column shows the associativity. Timing details are given in billion clock cycles. Although the timing is improved when cache size increases, the rate plateaus. In these experiments, we found that 8kB cache size with associativity of 2 is a suitable configuration.

Power consumption is estimated using the Tensilica processor design platform (memory is not included in the estimation). The 16 processor system takes 600 mW at clock frequency of 982 MHz. The area size for 16 processors system is 1.28 mm^2.

Table 7.1: Cache Performance Analysis. (clk. cycles)

Size (kB)	0	1	2	4	8	16	32
Asso.1	4.54	2.83	2.63	2.44	2.41	2.35	2.31
Asso.2	-	2.77	2.47	2.39	2.33	2.31	2.30
Asso.4	-	-	2.47	2.34	2.33	2.33	2.30

8 Conclusions and Future Work

In this paper, we have proposed an MPSoC based embedded solution for short read genome alignment. We introduced a ring pipelined architecture and partition reference genome to run "BWA-al" on embedded processors each with 2GB memory. Experimental results show a speed up of seven times for 16 processors can be achieved with the same accuracy as original "BWA-aln" on an Intel server.

In future work, this work will be extended to support long read genome alignment and different aligners. Other steps in the genome computational steps such as the variant caller will be ported to work on embedded processors.

Bibliography

[1] Xtensa Processor. Tensilica Inc. `https://ip.cadence.com/hwdes/`. Accessed: 2017-07-07.

[2] A. M. Aji, L. Zhang, and W. c. Feng. Gpu-rmap: Accelerating short-read mapping on graphics processors. In *13th IEEE CSE*, pages 168–175, Dec 2010.

[3] S. Aluru and N. Jammula. A review of hardware acceleration for computational genomics. *IEEE Design Test*, 31:19–30, Feb 2014.

[4] S. Canzar and S. L. Salzberg. Short read mapping: An algorithmic tour. *Proceedings of the IEEE*, 105(3):436–458, March 2017.

[5] Shaolong Chen and Miquel A. Senar. Accelerating bwa aligner using multistage data parallelization on multicore and manycore architectures. *Procedia Computer Science*, 80:2438 – 2442, 2016.

[6] E. B. Fernandez, W. A. Najjar, S. Lonardi, and J. Villarreal. Multithreaded fpga acceleration of dna sequence mapping. In *HPEC*, pages 1–6, Sept 2012.

[7] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.

[8] H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. Energy-efficient adaptive pipelined mpsocs for multimedia applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(5):663–676, May 2014.

[9] H. Javaid, D. Witono, and S. Parameswaran. Multi-mode pipelined mpsocs for streaming applications. In *18th ASP-DAC*, pages 231–236, Jan 2013.

[10] Haris Javaid, Muhammad Shafique, Sri Parameswaran, and Jörg Henkel. Low-power adaptive pipelined mpsocs for multimedia: An h.264 video encoder case study. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 1032–1037, New York, NY, USA, 2011. ACM.

[11] Petia Kovatcheva-Datchary and Tulika Arora. Nutrition, the gut microbiome and the metabolic syndrome. *Best Practice & Research Clinical Gastroenterology*, 27(1):59–72, 2013.

[12] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[13] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 25(14):1754, 2009.

[14] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078, 2009.

[15] Y. Liu and B. Schmidt. Cushaw2-gpu: Empowering faster gapped short-read alignment using gpu computing. *IEEE Design Test*, 31(1):31–39, Feb 2014.

[16] 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, 2012.

[17] Michael L Metzker. Sequencing technologies-the next generation. *Nature reviews genetics*, 11(1):31–46, 2010.

[18] Alexander S Mikheyev and Mandy MY Tin. A first look at the oxford nanopore minion sequencer. *Molecular ecology resources*, 14(6):1097–1102, 2014.

[19] Joshua Quick, Nicholas J Loman, Sophie Duraffour, Jared T Simpson, Ettore Severi, Lauren Cowley, Joseph Akoi Bore, Raymond Koundouno, Gytis Dudas, Amy Mikhail, et al. Real-time, portable genome sequencing for ebola surveillance. *Nature*, 530(7589):228–232, 2016.

[20] Steven L Salzberg and James A Yorke. Beware of mis-assembled genomes. *Bioinformatics*, 21(24):4320–4321, 2005.

[21] Shuji Suzuki, Masanori Kakuta, Takashi Ishida, and Yutaka Akiyama. Gpu-acceleration of sequence homology searches with database subsequence clustering. *PLoS one*, 11(8):e0157338, 2016.

[22] Yatish Turakhia, Kevin Jie Zheng, Gill Bejerano, and William J. Dally. Darwin: A hardware-acceleration framework for genomic sequence alignment. 2017.