PARALLEL ARCHITECTURE AND HARDWARE IMPLEMENTATION OF PRE-PROCESSOR AND POST-PROCESSOR FOR SEQUENCE ASSEMBLY

Yuan-Hsiang Kuo, Chun-Shen Liu, Yu-Cheng Li, Yi-Chang Lu

Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan

ABSTRACT

In current DNA sequence assembly flows, a time-consuming pre-processing routine is usually executed to remove low quality data for smaller problem sizes and better final results. The step usually takes 25%~90% of the total computation time. To speed up the process, based on the characteristics of DNA data, we propose to use digital processing hardware, with parallel and pipeline capabilities, to accelerate the pre-processing step. In our FPGA implementation, the computing speed can be improved up to 2,700 folds. With similar design concepts, we also implement a post-processor to convert assembly results from color space to base space whenever the step is necessary. Both designs enable faster sequence assembly, which will greatly benefit the research in genomics and related fields.

Index Terms—Bioinformatics, data pre-processing, parallel processing, pipeline processing, digital integrated circuits

1. INTRODUCTION

Over the past few years, reconstructing genome sequences from experiments is an important research topic in bioinformatics. Sequence alignment and sequence assembly are widely applied methodologies. The process begins by breaking DNA into a large amount of short fragments. Fragments generated by different shotgun sequencing technologies have different properties [1]. After shotgun sequencing, sequence alignment reconstructs original DNA sequence by mapping these fragments, or *reads*, to a reference DNA sequence. On the contrast, sequence assembly assembles reads without a reference. Due to the lack of reference sequences, assembly methods usually lead to uncertain results and cost much more time. Therefore, we focus on the improvement of sequence assembly speed in this paper.

As mentioned in [2], assembling reads into complete genome meets two challenges: errors in reads, and repeats in genome. Human mistakes during lab work or limitations in technology may cause error bases in each short read, which result in merging wrong reads. As for repeats, it is normal features in genome, but it's difficult to position these reads correctly.

Eulerian path algorithm is developed to solve these issues [3]. This graph-based approach breaks every read into continuous k-mers which are represented as nodes in the graph. Each k-mer is a substring of length k of the original read and linked by edges of the graph. Assembly tools search through k-mers and connect nodes to build directed De Bruijn graph. Using this approach, assembly problem corresponds to finding a path through all edges. In theory, the complexity of Eulerian path problem is linear time [2]. In all of Eulerian path tools, Velvet [4] provides a mature processing flow, therefore, we use it for our designs and experiments.

In Velvet assembly flow (Fig. 1), the pre-processor filters out low quality reads and, whenever necessary, converts data format into the one that Velvet program can recognize. Then Velvet program assembles reads to contiguous sequences (Contig). If the original data are coded in color-space, the post-processor is required to convert the Velvet outputs to the final genome sequence. In our experiments, we use different datasets to record the computation time of each stage. Pre-processing usually takes about 25% to 90% (38% and 22% in the cases below) of the total time, which is proportional to the number of reads. As to the post-processing time, it depends more on the contig numbers generated after the Velvet stage.

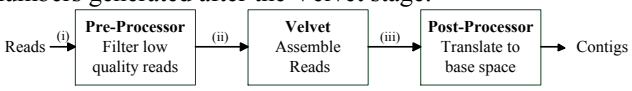

Fig. 1 Velvet assembly flow

	Read No. (i)	Pre-Proc. time	Read No. (ii)	Velvet time	Contig No. (iii)	Post-Proc. time
TB Paired	148M	4h 35m	53M	2h 48m	8.6k	4h 43m
TB Single	74M	2h 13m	60M	3h 33m	23k	4h 18m

TB: MDR4_F3, MDR4_R3 Read length: 50

Table 1 Simulation result of velvet assembly flow

Table 1 shows two examples using high coverage color-space Tuberculosis (TB) data. From the results, we find that if we can greatly shorten the computation time of either stage, the total assembly time will be reduced significantly,

too. Because of the characteristics of reads, we propose to use digital processing hardware, with parallel and pipeline capabilities, to accelerate the pre-processing and post-processing stages.

2. PRE-PROCESSOR ARCHITECTURE

The main function of the pre-processor is to filter out low quality reads. Reads can be coded in either base-space or color-space, depending on the technology that the next generation sequencing (NGS) machine uses. Since some assembly tools can only recognize data in base-space, the pre-processor will convert color-space data into pseudo-base-space data for later processing. In such a case, we will need a post-processor to convert the data in pseudo-base-space back to color-space.

In these data, every read consists of two major fields: one is the sequence of base or color, and the other is the quality value (QV) corresponding to each spot on the sequence. A base sequence is coded in ATCG, while a color sequence is coded with numbers obtained from two neighboring bases as shown in Fig. 2 [5][6]. As to the QV, it is calculated using the following equation:

$$Q = -10\log_{10} P, \quad (1)$$

where Q is the quality value and P is the predicted probability that the base/color call is incorrect.

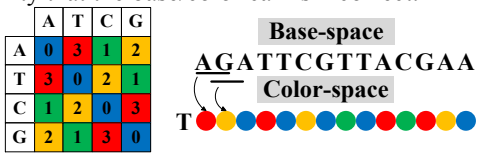

Fig. 2 Color coding in SOLiD systems

Since low quality base data usually leads to misalignment when building the De Bruijn graph. A pre-processing routine is always used to removes low quality reads through checking the quality values. The four major filtering rules are summarized in Table 2.

Parameter	Description
MQV_{min}	Minimum acceptable median quality value
L_{min}	Minimum acceptable length of read
N_{lq}	Upper bound number of low QV
F_{best}	Best read fraction

Table 2 Filtering rules of the pre-processor

Based on the rules above, we propose a pre-processor hardware design as shown in Fig. 3. This design includes two parts: a set of filter chips and a master chip. Filter chips calculate the properties of fragments (such as read length, median QV of every read), and remove bad reads. Filtering operation for each read can be conducted in parallel, so we separate all reads into several groups and assign the groups to different filter chips. By doing so, filter chips process reads in parallel thus decrease the execution time. The master chip analyzes median QV data generated by filter chips. The number of filter chips connected to the master chip is of users' choice. The number recommended is equal to the longest read length based on the hardware architecture.

In conventional software approaches, programs process the properties of reads sequentially, so it costs lots of time. However, in our hardware design, all units in the pre-processor are running simultaneously in a pipeline manner as shown in Fig. 4. Therefore, the execution time can be greatly reduced.

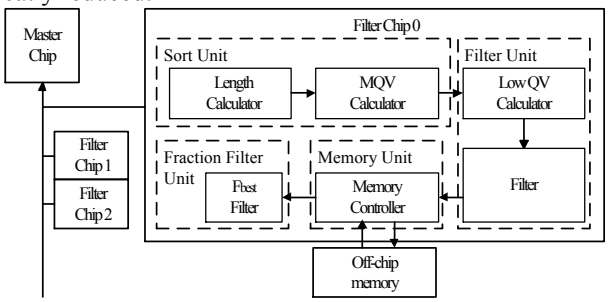

Fig. 3 Pre-processor architecture

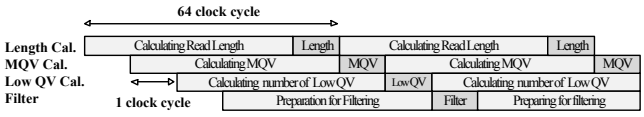

Fig. 4 Pipeline architecture of the pre-processor

Filter chip is composed of four units: *Sort* unit, *Filter* unit, *Memory* unit and *Fraction filter* unit. *Sort* unit is in charge of calculating the length of reads and the median of QVs. Since the data of a read arrive in serial, the length is determined when the last base/color comes in. As to the median computation, the unit first sorts the QVs and pick the numbers in the middle. In theory, the complexity of sorting is O(*NlogN*). However, because of the parallel property, hardware can sort reads in linear time. The *Sort* unit can figure out the median QV and read length as soon as the end of a read comes in. Then the two numbers are transmitted to the *Filter* unit and the master chip.

The *Filter* unit has two main functions. First, reads failed to comply with the three filtering rules, MQV_{min}, L_{min}, and N_{lq}, will be detected by comparators and counters in the unit. Second, for color-space inputs, the *Filter* unit coverts data into pseudo-base-space by removing the first two color calls and mapping {0,1,2,3} directly to {A,C,T,G} (instead of decoding the number to one of its four possible sets).

Furthermore, a read could be a single segment or paired segments. For paired reads, there are two different forms: paired-end (F5 and R3) reads and mate-paired (F3 and R3) reads. Since some of the assembly tools, like Velvet, recognize paired-end data only, the *Filter* unit must change mate-paired data into paired-end reads by reversing the F3 sequences.

The *Memory* unit and *Fraction filter* unit are designed for the F_{best} criterion in Table 2. If we want to choose the best part of the reads, the median QVs of all reads should be counted by the master chip first. For this reason, reads

should be stored on off-chip memory temporarily until the master chip finishes counting. Then *Fraction filter* unit removes more reads according to the statistics. Then the reads are ready for the assembly tools.

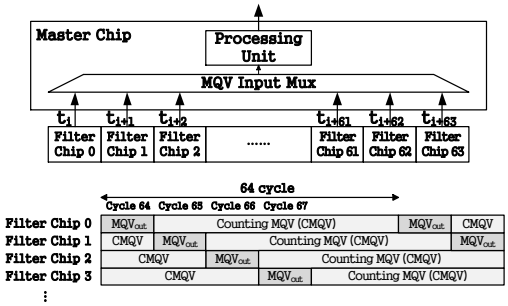

Fig. 5 Configuration and timing schedule of the master chip

Fig. 5 shows how the master chip functions. Master chip consists of counters which record the histogram of quality values reported by *N* filters (*N*=64 in Fig. 5). Each filter chip outputs its median QV every *N* clock cycles. In our design, the output timing is shifted by one clock in each filter, so the master chip can receive one median QV at a time. With this technique, the filter chips will not generate outputs at the same time, which greatly reduces the number of adders in the master chip. Another advantage of this design is that the master chip architecture is scalable. If the maximum length of the reads is increased to 128, we can have the master chip connected to 128 filter chips. Because of the pipeline architecture, the computation time remains the same, and it is proportional to the number of total reads. We can further improve the computing speed by cascading master chips (and adders).

3. POST-PROCESSOR ARCHITECTURE

If we convert color-space data into pseudo-base-space data for Velvet inputs, we will need the post-processor to convert the Velvet outputs from pseudo-base-space to the final genome sequence. Mapping pseudo-base-space data to color-space data is straight forward. However, color-space outputs cannot be translated to a genome sequence without the information of the first nucleotide. Therefore, most post-processing routines have to trace all reads and align them all over again according to the position information provided by Velvet. Since there are numerous reads, the computing time is very long. Besides, those algorithms ignore the fact that Velvet has provided an assembled sequence in pseudo-base-space. Therefore, by sacrificing a little bit accuracy, we purpose a faster method as shown in Fig. 6.

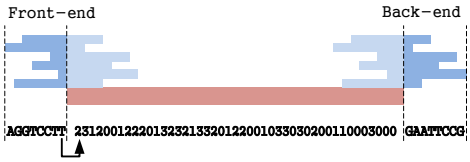

Fig. 6 Concept of post-processor

We first map the Velvet sequence results from pseudo-base-space to color-space. Then we align only first *M* reads in the front-end part to determine the first nucleotide of the assembled color-space sequence. With this information, we can change the assembled sequence from color space to base space. It is not necessary to align all the reads, so the execution time will decrease a lot. Using similar concepts, the reads in the back-end are also aligned to extend the length of the assembled sequence. According to our experiments, the accuracy of the results improves as the number of *M* increases. In our design, we choose *M*=10, and the accuracy is 92%. If we increase *M* to 20, the accuracy is above 95%.

Fig. 7 shows the architecture of our post-processor. The *Post-filter* unit receives assembled sequences data and their corresponding reads from Velvet. Based on the position information, the *Post-filter* unit uses two sorters to find the first and last ten reads for alignment purposes. In the filtering process, assembled color-space sequences are store in the *Memory* unit which includes two SRAMs. Then the *Alignment* unit decodes output reads found by the *Post-filter* unit and merges them with the assembled sequences from the SRAMs. As Fig. 8 shows, *Post-filter* unit and *Alignment* unit work in pipeline to improve the timing, where two SRAMs are used to make sure the data are stored and updated efficiently.

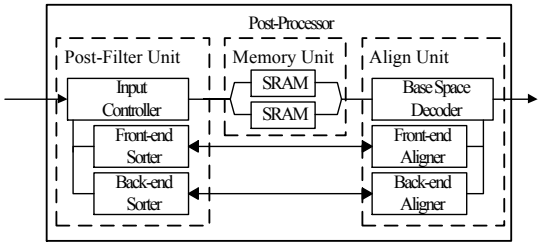

Fig. 7 Post-processor architecture

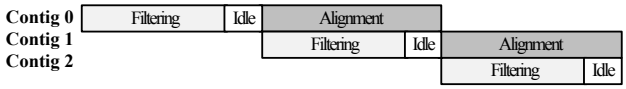

Fig. 8 Pipeline of post-processor

4. RESULTS

We implement our pre-processor chip with TSMC 90 nm technology. The filter chip and master chip are designed so that the master chip can communicate with 16 filter chips, where the maximal number of the connected filter chips is limited by IO pads. The specifications of the chipset are shown in Table 3.

Chip	Filter Chip	Master Chip
Clock frequency	100 MHz	
Chip Area	1.796 mm^2	2.265 mm^2
PAD	123	145
Power	30.89 mW	21.76 mW

Table 3 Specifications of the pre-processor chipset

	Software	Hardware
Proc. Time	4hr35min	5.92sec

Table 4 Execution time of pre-processor in software and hardware without data transmission time

Table 4 shows the execution time of the pre-processing step using a real TB genome dataset as an example (148M reads, Tuberculosis DNA 138000463_20110104_2_MDR4). From Table 4, we can see that the computation speed of the hardware design is much faster the software version. We also design a larger pre-processor which integrates 4 filter chips and 1 master chip into a single chip, which could further improve the processing speed but at a higher manufacturing cost.

In addition to the ASIC design summarized in Table 3 and Table 4, we also implement our pre-processor on FPGA boards. We choose to use two DE4 boards because one DE4 board won't be able to store all the data required for our TB test case. Table 5 shows the execution time as different I/O interfaces and numbers of DE4 FPGA boards are used. Because the pre-processors on FPGAs start to transmit processed data while processing the incoming data, the transmission time between FPGAs and the CPU will not affect the total execution time until the processing time is shorter than the transmission time bounded by the channel bandwidth.

# of DE4	SATA 3.0		PCI-E x8	
	without faction filter	with fraction filter	without faction filter	with fraction filter
2	47.36s	94.72s	47.36s	94.72s
4	23.68s	47.36s	27.68s	47.36s
8	23s	23.68s	27.68s	27.68s
16	23s	23s	27.68s	27.68s

Table 5 Execution time of pre-processing with different numbers of FPGA boards and I/O interfaces

We also implement a post-processor chip with TSMC 90 nm technology. The specifications of chips are shown in Table 6. We also compare the hardware processing time with the software version in Table 7. Although there is a trade-off between the speed and accuracy (8% in this case) when implementing our hardware version as explained in Section 3, the speed improvement is more than 30,000 times.

Chip	Post-Processor
Clock frequency	100 MHz
Chip Area	1.416 mm^2
PAD	103
Power	50.7 mW

Table 6 Post-processor implementation results

	Software	Hardware
Proc. time	4hr43min	0.49sec

Table 7 Execution time of post-processor in software and hardware

5. CONCLUSION

We propose to use pre-processor and post-processor hardware to accelerate DNA sequence assembly processes in this paper. The design of the pre-processor and post-processor adopts pipeline techniques to take the full advantages of using hardware. Compared to conventional software approaches, the pre-processor increases the computation speed by over 2,700 times. Moreover, the architecture of the pre-processor is scalable. We also propose a new post-processing algorithm to improve the efficiency and the computation speed with a small trade-off in accuracy. Both designs accelerate sequence assembly processes, which will make sequence assembly much faster than they used to be.

6. ACKNOWLEDGEMENT

This work is partially supported by National Science Council, Taiwan, under Grant number NSC 99-2221-E-002-216-MY3. The authors would like to thank Dr. Ker-Chau Li, Dr. Shin-Sheng Yuan, and Dr. Hsuan-Yu Chen, all with Institute of Statistical Science, Academia Sinica, for providing the NGS data and valuable comments.

7. REFERENCES

[1] M. L. Metzker, "Sequencing technologies - the next generation," *Nat Rev Genet,* vol. 11, pp. 31-46, 2010.

[2] M. Pop, S. L. Salzberg, and M. Shumway, "Genome sequence assembly: algorithms and issues," *Computer,* vol. 35, pp. 47-54, 2002.

[3] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences,* vol. 98, pp. 9748-9753, August 14, 2001 2001.

[4] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research,* vol. 18, pp. 821-829, May 1, 2008 2008.

[5] K. J. McKernan, *et. al.*, "Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two base encoding," *Genome Research,* June 22, 2009 2009.

[6] H. Breu, "A theoretical understanding of 2 base color codes and its application to annotation, error detection, and error correction," *White Paper SOLiD System* Applied Biosystems, 2010. Available: http://www3.appliedbiosystems.com/cms/groups/mcb_marketing/documents/generaldocuments/cms_058265.pdf