FAssem : FPGA based Acceleration of De Novo Genome Assembly

Sharat Chandra Varma, Paul Kolin, M. Balakrishnan, Dominique Lavenier

FAssem : FPGA based Acceleration of *De Novo* Genome Assembly

B. Sharat Chandra Varma, Kolin Paul, M. Balakrishnan
Department of Computer Science and Engineering
Indian Institute of Technology Delhi, New Delhi, India
Email: {varma, kolin, mbala} @cse.iitd.ac.in

Dominique Lavenier
IRISA / INRIA
35042 Rennes, France
Email: lavenier@irisa.fr

***Abstract*—Next generation sequencing technologies produce large amounts of data at very low cost. They produce short reads of DNA fragments. These fragments have many overlaps, lots of repeats and may also include sequencing errors. The assembly process involves merging these sequences to form the original sequences. In recent years many software programs have been developed for this purpose. All of them take significant amount of time to execute. Velvet is a commonly used *de novo* assembly program. We propose a method to reduce the overall time for assembly by using pre-processing of the short read data on FPGAs and processing its output using Velvet. We show significant speed-ups with slight or no compromise on the quality of the assembled output.**

***Keywords*-FPGA; Acceleration; Next Generation Sequencing Assembly; Bioinformatics;**

I. Introduction

Next-generation sequencing (NGS) platforms generate enormous amount of data at very low cost at much greater speeds when compared to older sequencing platforms. This has opened up various opportunities for biologists to analyze this data for various purposes including design of personalized drugs.

A complete set of all genes along with non-coding DNA in an organism is called a ***genome***. The NGS machines generate short fragments called ***"reads"*** of length thirty five to few hundreds of base-pairs. These reads are part of a large genome containing millions of base-pairs (the size ¿ of the human genome = 3x109 bp). ***Sequence Assembly*** is a computational biology problem where the genome is assembled using the reads generated from the NGS machine. The construction of genome is more complex than the well studied shortest super-string construction problem. The problem becomes more computationally intensive as the sequencing machine generates reads with errors.The complexity of problem further increases due to repeats which are some common regions (sequences) in the genome. Also NGS machines have a constraint on the length of the reads generated. If the length of the read is less than the repeat, identifying the portion of the genome from where the read came from becomes very difficult and is practically impossible to solve.

De novo assembly in software is done using one of the two graph based methods- the overlap layout consensus (OLC) and the de Bruijn. In this paper, we propose a hybrid approach where we generate the overlap and layout using the OLC technique and build the consensus using a de Bruijn method. The key innovation is to use a (parallel) hardware implementation to remove the 'redundancy' in the input read data and using state of the art highly computationally intensive de Bruijn based Velvet software [1] to build the consensus sequence. Even though *de novo* assembly is slower than mapping based assembly, it has its advantages and is widely used by bioinformaticians. We attempt to accelerate it using FPGA based accelerators. We estimate speed-ups of 13x using our approach.

The main contributions of this paper are the following

1) Novel way to speed-up *de novo* assembly of NGS data using FPGAs.
2) Hardware-software co-design to achieve this.
3) Efficient implementation of hardware on FPGA.

Section II describes the work done by other research groups that have been reported in the literature. In section III we describe the overall approach and high level implementation used for initial analysis and to do feasibility study. This study leads us to prepare an overall methodology to achieve speed-ups using FPGAs which is described in section IV. Section V shows some of the results and this is followed by conclusion in section VI.

II. Related Work

Several groups have attempted to accelerate NGS short read mapping using FPGAs. Corey B. Olson et al. [2] has shown acceleration of short read mapping on FPGA. The authors compare their results with BFAST software [3] and show 250x improvement and 31x when compared to Bowtie [4]. Edward Fernandez et al. [5] and O. Knodel et al.[6] have also accelerated NGS short read mapping. The Convey GraphConstructor (CGC) use FPGAs to accelerate *de novo* assembly and show speedups of 2.2x to 8.4x [7].

III. FAssem Assembly

We chose to accelerate de Bruijn based assembly, as they take less amount of time to execute when compared to OLC based assemblers. We use the fact that there is a lot of redundancy in short read sequencing. The de Bruijn based software assemblers takes several GBs of RAM space while executing [8]. Efficient implementation of de

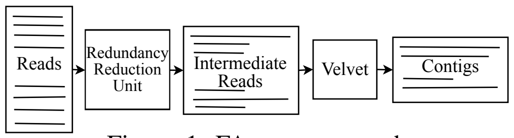


Figure 1: FAssem approach

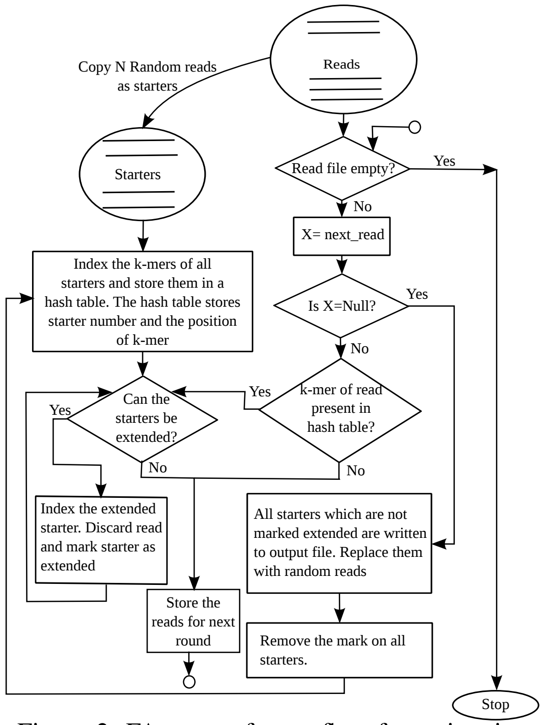

Reads
Copy N Random reads as starters
Starters
Read file empty?
Yes
No
X= next_read
Index the k-mers of all starters and store them in a hash table. The hash table stores starter number and the position of k-mer
Is X=Null?
Yes
No
Can the starters be extended?
Yes
k-mer of read present in hash table?
Yes
No
No
Index the extended starter. Discard read and mark starter as extended
All starters which are not marked extended are written to output file. Replace them with random reads
Store the reads for next round
Remove the mark on all starters.
Stop

Figure 2: FAssem software flow for estimation

Bruijn based assemblers on an FPGA is difficult due to the memory resource constraints in current FPGAs. We model a hybrid approach where we implement a part of OLC based assemblers on FPGA to remove the redundancy present in the reads. We run the de Bruijn based graph assembler in software on the reduced set of reads from the FPGA.

This approach allows us to effectively use the FPGA resources for removing redundancy in the reads. A high level design of our approach is shown in Fig. 1. The reads are passed through Redundancy Remover Unit (RRU) implemented in FPGA, which acts as preprocessor. The output from the RRU is given to Velvet for constructing the final contigs.

Our idea is based on an open source software known as Mapsembler [9]. We used this software to study the benefits of our approach. Mapsembler is a software which does targeted assembly. It takes NGS raw reads and a set of input sequences (starters). The software determines if the starter is read coherent, i.e. starter is a part of the original sequence. The neighborhood of the starter is given as output if the starter is read coherent. All the k-mers in all of the starters are hashed and stored in a hash table. The hash table consists of starter number and the corresponding position of the k-mer in that particular starter. A read is taken from the NGS read set and the respective k-mers are searched in the hash table. If the k-mer is already hashed, the corresponding starters are tried for extension with the reads.

The main thrust in our approach is to find the overlap region between reads and store the overlap region only once. The overall flow diagram is as shown in Fig. 2. We do a streaming design where processing elements are connected in series. We consider N processing elements. 'n' random reads are stored as starters. A read is taken from the remaining read set and checked if it can extend any of the starters starting from the first starter. If the read can extend the starter, the starter is extended and the read is deleted. If the read does not extend the starter, the read is stored in a new read set. We use the term ***"round"*** frequently in the rest of the paper which means that all the reads from the read set are compared once with current set of starters and tried for extensions. After all the reads are exhausted in the "extension" process, the starters which could not be extended are stored in the output file. These starters are replaced by new random reads in the next round.

This process of reducing the redundancy in the new readset is repeated with the new read set and replaced starters. This redundancy reduction is repeated for several rounds till the number of reads is less than the starters. The output file now contains intermediate reads of longer length from which redundant information has been removed. The output file is given to Velvet software for generating contigs and removing errors.

IV. Hardware Implementation

We implement the RRU in FPGA. The hardware model differs from the software model significantly. In the hardware model, we do not implement the hash based searching of k-mers due to constrained memory resources. The model we propose is equivalent to the software model in terms of other functions like extending. In this model the reads are compared with the starter and tried for extension at its ends. In each cycle the read is shifted and checked if it can extend the starter. The extension phase is expensive as the number of cycles needed for extension is equivalent to the difference of read length and k-mer length. From the software implementation, it is observed that for a single round, the number of reads used for extension are very small when compared to reads that extend the starters. To take advantage of this feature, we propose a *pre-filter* block. The pre-filter is added before the extension phase. The pre-filter filters the probable reads that might not extend the starter and pass only the reads that might extend the starter. Pre-filter compares the signature of the reads with the signature of the starter. This Signature is called the ***ReadVector*** and is constructed by encoding the 4-mers in binary format. 4-mer was chosen for signature because the vector width would be 256 corresponding to 4^4. If we choose a signature with more than 4-mer, then the signature will

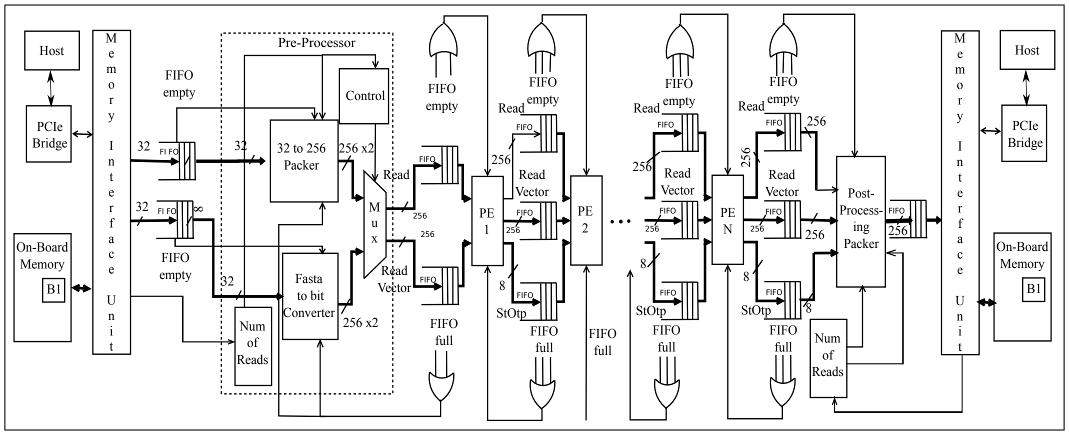


Figure 3: Block diagram of hardware implementation.

become much more lengthy and hence would require large memory for its storage and larger amount of resources for doing the pre-filter. For example, the readvector for read AAAAAAAGGGGG is "100100...001".

We use AlphaData board for hardware implementation [10]. The board has a PCIe bridge which is used for data transmission between host and vice versa. The Memory interface unit connects the on-board memory and the PCIe bridge. In our design we use the *"Memory Interface"* unit to send data to a *"Pre-Processor"*. From the pre-processor a series of *"Processing Elements"* (PEs) are connected through a set of fifos. The last PE is connected to a *"Post Processor"* connected back to memory interface unit. The expanded diagram showing different stages is shown in Fig. 3. The Memory interface unit and PCIe bridge are shown twice both at left and right to simplify the diagram, but refers to the same unit.

The read set in fasta input file format is sent from the host to the FPGA board through PCIe bus. In order to remove communication delay between host and FPGA board, we consider the double buffering model, where the data is buffered in two buffers. One of the buffers is used for transferring data from the host, while the other will be used for processing the data. Similarly, we use double buffering at the output too. For initializing the starters, we encode the most significant three bits of the read. The fourth bit is used for marking the read that it has extended as a starter. In order to reconstruct the starters, the starter and the position of the extension is sent as output through the fifoSet. Reconstruction of starters is done in software.

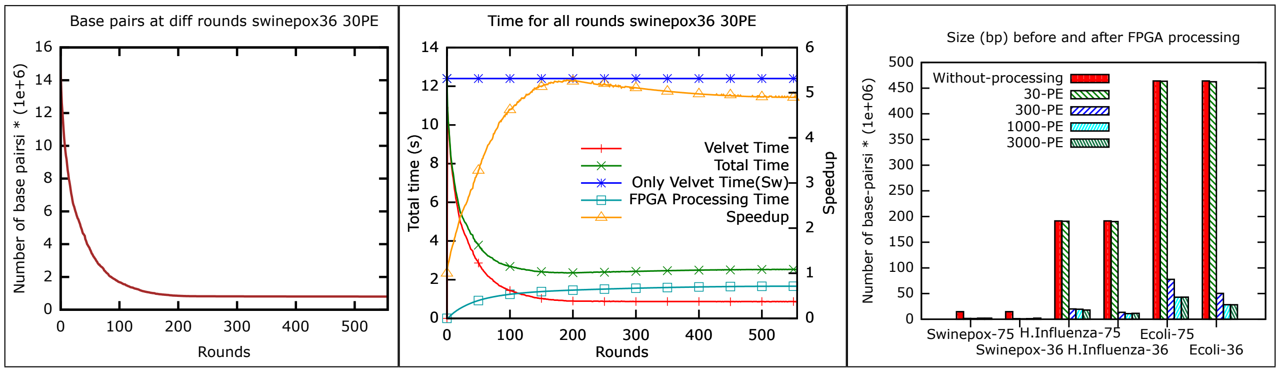
V. Results and Discussion

Zhang et al. [8] have done a comparison of *de novo* assembly softwares. The authors have provided scripts for generating the read-set from the genome. We used these scripts to generate the read files for ecoli, swinepox and human influenza. For evaluating our approach, we generated the single ended readset with 100x coverage for read-length 36 and 75 and 1% error rate similar to what was reported by Zhang et al. [8]. For the software only flow time, Velvet software was run using the readset directly on a desktop computer with Intel (R) Core (TM) 2 Duo CPU E4700 running at 2.60GHz with 4GB RAM. We have implemented the RRU on FPGA and obtained the clock period and utilized FPGA resources after running place and route tools provided by Xilinx ISE 14.1 [11]. We use these parameters to estimate the speed-ups for running the Velvet on the output of RRU after each round. From place and route tools, the maximum clock frequency for the whole of the design was found to be 200 MHz. We get better performance by using multiple clocks. The sequence coder and generate units were able to run at a maximum frequency of 350 MHz on Virtex-6 FPGA. The maximum frequency of operation for the rest of the units was 200 MHz. A total of 15 PEs could be implemented on Xilinx Virtex-6 XC6VLX130T FPGA. Note that for design with larger number of PEs where multiple FPGAs would be required, we have not considered inter FPGA transfer time in our estimates. We assume FPGAs are connected in series and the data is streamed from the host, through the FPGAs and finally, back to the host.

Table I: Maximum speed-ups over software.

Sample	Swinepox	Swinepox	H. Influenza	H. Influenza	Ecoli	Ecoli
Read-length	75	36	75	36	75	36
PE \Size	30.8 MB	48.4 MB	449 MB	729.7 MB	1.2 GB	2.1 GB
30 PE	3.5x	5.2x	1.1x	1.09x	1.2x	1.09x
300 PE	13x	11.9x	3.2x	3.6x	2.5x	2.1x
1000 PE	6.5x	10.5x	6.8x	6.0x	4.4x	5.02x
3000 PE	4.8x	7x	6.8x	10x	6.5x	9.2x

Fig. 4a and 4b show the graphs of the speed-ups at different rounds for swinepox with read length 36 using 30 PE. We have considered the worst case time by setting the threshold value for the pre-filter to zero. Similarly, we have the results on ecoli, swinepox and human influenza, not shown here due to space constraints. We observe same



(a) Input Size using 30 PE for swinepox read-length 36.
(b) Speedups using 30 PE for swinepox read-length 36.
(c) Input sizes (bp) before and after FPGA processing.

Figure 4: Consolidated results

trends. The reduction in size of the input file in terms of base-pairs to Velvet software is shown in Fig. 4c. For a larger genome like ecoli, we need to have more processing elements to get significant speed-ups. The maximum speed-ups are tabulated in Table I. The speed-ups in each case first increases, reaches a peak and then tapers down. The initial increase can be attributed to the high reduction of input file size during the initial rounds. After these initial rounds, the redundancy removal is more limited and so the time taken by Velvet is almost constant. The FPGA processing time is incremental in nature and hence goes on increasing after each round. Even though there is not much redundancy removal during the later rounds, the hardware unit takes at least as many cycles as the number of reads and writes in each PE.

The most popular metrics to measure quality are the maximum length of the contigs and the ***"N50"***. N50 is the minimum length of the contig such that summing up the length of only those contigs whose length is more than N50 cover 50% of the genome. From the various experiments conducted we observed that by not allowing mismatches during extension, there was no (significant) loss in quality of output as shown in results.

VI. Conclusion

NGS sequence assembly is becoming very important as it has opened many opportunities in personalized medicine, meta-genomics, etc. *De novo* assembly has its own advantages over comparative mapping based assembly, but take more time to execute. We have come up with an hybrid approach which uses techniques from OLC method and de Bruijn method for accelerating assembly using FPGAs. From the results we find that the speed-up is dependent on the nature and size of input data. For a fixed number of PEs, the speed-up first increases and then tapers down with larger number of rounds as FPGA processing time starts dominating. Maximum speed-up increases with number of PEs and reduces after reaching peak. We estimate speed-ups up-to 13x using our hybrid approach.

References

[1] D. R. Zerbino and E. Birney, "Velvet: Algorithms for De Novo Short Read Assembly using De Bruijn Graphs," *Genome Research*, vol. 18, no. 5, pp. 821–829, 2008.

[2] C. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. Ruzzo, "Hardware Acceleration of Short Read Mapping," in *IEEE Symposium on FCCM*, May 2012, pp. 161 –168.

[3] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: An Alignment Tool for Large Scale Genome Resequencing," *PLoS ONE*, vol. 4, Nov 2009.

[4] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, 2009.

[5] E. Fernandez, W. Najjar, E. Harris, and S. Lonardi, "Exploration of Short Reads Genome Mapping in Hardware," in *International Conference on FPL*, Sept 2010, pp. 360 –363.

[6] O. Knodel, T. Preusser, and R. Spallek, "Next-generation Massively Parallel Short-Read Mapping on FPGAs," in *IEEE International Conference on ASAP*, sept. 2011, pp. 195 –201.

[7] Convey Computer, "Convey GraphConstructor," www.conveycomputer.com.

[8] W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, and B. Shen, "A Practical Comparison of De Novo Genome Assembly Software Tools for Next-Generation Sequencing Technologies," *PLoS ONE*, vol. 6, no. 3, March 2011.

[9] P. Peterlongo and R. Chikhi, "Mapsembler, Targeted and Micro Assembly of Large NGS Datasets on a Desktop Computer," *BMC Bioinformatics*, vol. 13, no. 1, 2012.

[10] Alpha-Data, "Alpha-Data FPGA Boards," www.alpha-data.com/.

[11] Xilinx, "Xilinx FPGAs, ISE," www.xilinx.com.