Hardware Acceleration of Optically Labeled Human Genome Sequencing using a Novel Algorithm

Karishma Shiraj Mulani, Harish Kumar, Gaurav M.K and Sumam David S.
Department of Electronics and Communication Engineering
National Institute of Technology Karnataka, Surathkal, Mangalore-575025, India
Phone: +91-9741368967, Email: karishma.mulani@utexas.edu

***Abstract*—Recently, reconstruction of the entire DNA sequence from optically labeled genomes has been explored. In this paper, we present details of a novel algorithm for this genome assembly. We elucidate the design methodology and results for a multi-core CPU (1.98x speedup) and FPGA (7.022x speedup) implementation to accelerate the computations.**

***Keywords*-Genome assembly; Multi-core CPU; FPGA; Acceleration; Parallel algorithm**

I. Introduction

Genome sequencing is the process of assembling a complete genome from fragmented reads. *De Novo* based assembly is carried out in the absence of a reference. Over the years, several techniques have been developed using short reads for the same. But short read techniques are limited in the sense that they cannot be used to reconstruct large genomes [1]. To overcome the limitations of short read techniques, a novel optical label based approach was proposed. P. Meng *et al.* [2] present implementations to accelerate the large-scale optical label based genome assembly using GPUs and FPGAs. The algorithm adopted by them is based on a statistical model proposed by Valouev [3] and the design utilizes constants derived from [3] which are tuned to suit their experiment input data - a synthetic human genome.

P.Meng *et al*. [2] opines that the Smith-Waterman algorithm [4] is not suitable for optically labeled genome sequencing as it takes into account only the immediate neighbours. We propose an algorithm based on the Smith-Waterman algorithm with an altered scoring scheme. An additional stage to gauge the similarity between neighbouring values has been added in our implementation. Parallelism can be exploited while implementing the Smith-Waterman Algorithm on the FPGA as proposed by Liao *et al*. in [5]. Khan *et al*. propose a multi-core system design [6]. This implementation reduces the time complexity by a factor of *N* as compared to the design in [2]. Furthermore, it is portable to any kind of genome and does not make use of constants specific to human genome. The techniques employed also reduce the overall complexity of the computations.

II. Algorithm

Since we deal with optical labels, the dataset comprises of arrays of floating-point values which represent the distance between optical labels in kilobase pairs (kB). These arrays are aligned using a dynamic programming algorithm. The algorithm consists of two parts:

1) Global alignment which compares each array with other arrays in the dataset.
2) Local alignment using a modified version of Smith-Waterman algorithm

If we have *F* fragments of DNA to be aligned, we need to compute a score between every pair of fragments in order to choose the best alignment. Smith-Waterman algorithm is the most accurate of all local alignment algorithms. It aligns DNA sequences (in our case the floating point arrays) using a similarity matrix. Using this matrix, we calculate the best score and indices to align.

We associate a scoring matrix *H* of dimensions $N \times M$ for the alignment of the arrays *X* and *Y* of sizes *M* and *N* respectively. Here, arrays *X* and *Y* are filled with floating-point values indicating the distances between each optical label as discussed before. The values filled in the $(i,j)^{th}$ cells of the matrix *H* are computed using the following formulae,

$$H(i,0) = 0, 0 \leq i \leq N \tag{1}$$

$$H(0,j) = 0, 0 \leq j \leq M \tag{2}$$

$$H(i,j) = max \left\{ \begin{array}{c} 0 \\ \text{H(i-1 , j-1) + s}(x_i, y_j) \\ \text{H(i-1 , j) - W} \\ \text{H(i , j-1) - W} \end{array} \right\}, \quad 0 \leq i \leq N, 0 \leq j \leq M \tag{3}$$

Here, $s(x_i, y_j)$ is the similarity function which is assigned +2 if a match is found or -1 if it is a mismatch. W is the gap scoring scheme which takes the value of 2.

$$s(x,y) = \left\{ \begin{array}{l} \text{+2 if x = y} \\ \text{-1 if x} \neq \text{y} \end{array} \right\}, W = +2 \tag{4}$$

Using the calculation of cell shown above, a sample scoring matrix is constructed using the formulae mentioned. Thus, the computation of the scoring matrix remains the same as that of the original algorithm in [4]. The novelty comes in the computation of the best score and the indices which align the two arrays *X* and *Y*. The best score which

978-1-5386-1112-8/18/$31.00 ©2018 IEEE

represents how well the two arrays under consideration overlap is obtained by taking the maximum value in the scoring matrix.

The best score is the maximum value in the score matrix. It is determined by iterating through the score matrix. Initially, we assigned the indices for alignment (*i_best* and *j_best*) to be the indices of the first non-zero cell in the score matrix. The (i,j)th indices correspond to fact that the ith element of array *N* and jth element of array *M* have to be overlapped. However, this method of assigning indices fails in case of *repeats*. Here, a repeat is defined as repeated values in two fragments which are not to be overlapped. To rectify this, we consider the matching of neighbouring cell values. Suppose *n* is the highest number of consecutive values that are repeating in a repeat, then indices are assigned only after *n+1* continuously increasing diagonal values are encountered in the score matrix. Otherwise, the indices are considered invalid for alignment(i_best=-1, j_best=-1). In a corner case scenario, when there are less than *n+1* values left and indices have not been assigned, indices are assigned after ensuring that the remaining diagonal values have a continuous increase in their cell values. If this does not hold true, then the indices are invalid. The modified scheme for indices assignment is employed only when there are a large number of continuous repeats.

III. Implementation

With the profiling tool *callgrind* in Valgrind, we were able to obtain the average duration of function calls in the C implementation of the genome assembly. The fraction of time utilization by the key functions has been listed in Table I. It can be inferred that the most computationally expensive process is the generation of the score matrix for the different pairs of DNA fragments. Hence, this segment of the system has been accelerated on the hardware. Since the backtracking algorithm has very little scope to be parallelized, it is executed on the PC. The values communicated from the FPGA back to the PC are the best score and the indices for alignment which are calculated on the FPGA. The overall hardware-software partitioning can be seen in Fig. 1.

Table I
Software Profiling Summary

Function	Description	Time Utilization(%)
score_matrix	Calculation of the score matrix and finding the best score along with the indices for alignment.	97.95%
alignment	Employing backtracking algorithm to reconstruct entire DNA.	0.39%
update_backtrace	Inserting results from score_matrix appropriately in *backtrack* array.	0.01%

To achieve a greater speed-up in the local alignment using the Smith-Waterman algorithm, we compute the values of

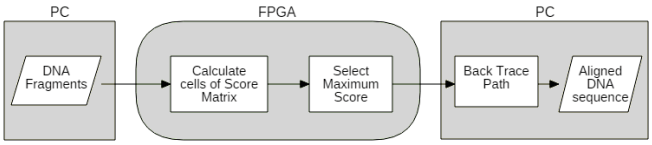

Figure 1. Hardware-Software Partitioning

cells in the score matrix in parallel. From the formulae mentioned in II, we see that each cell in the score matrix depends on three other cells. However, as pointed out by [5] and [6], we see that the elements in anti-diagonals of the scoring matrix can be calculated simultaneously provided the previous anti-diagonal elements are filled. This type of *wavefront propagation* method has been illustrated in Fig. 2. It reduces the time complexity significantly. Hence, we go sequentially from one anti-diagonal to another and calculate all the elements in each anti-diagonal simultaneously. This reduces the complexity of computation from *O(mn)* to *O(m+n)* which represents the reduction from number of cells in the score matrix comparing two arrays of lengths *M* and *N* to the number of anti-diagonals in this score matrix.

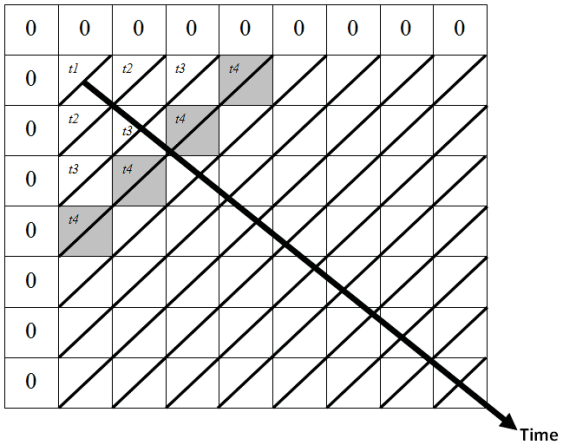

Figure 2. Parallel Implementation of Smith-Waterman Score Matrix

Therefore, the levels of parallelism identified from the algorithm are:

1) Alignment of multiple pairs of fragments of input data. (Level 1)
2) Computation of multiple elements in the score matrix which have no dependencies. (Level 2)

Additionally, for the Smith-Waterman algorithm with the modified index calculation scheme, we have one more level of parallelism where the neighboring values are compared.

If we have *F* fragments with an average length of *N* in the assembly process, the overall time complexity of this task is $O(F^2N)$. This is a reduced complexity by a factor of *N* as compared to the implementation mentioned in [2].

A. Multi-core CPU

The multi-core CPU design was implemented as a C program using OpenMP directives. In the local alignment using Smith-Waterman (SW) algorithm, the anti-diagonals need to be executed in a sequential manner. However, elements within the anti-diagonal can be computed in parallel. Hence, each CPU core is given an element to compute. Since these need not be synchronized, the CPU core can move onto the next set without any synchronization issues. This local alignment constitutes of a fine-grained parallelism. For global alignment, we compute the scores for each pair in parallel leading to a coarse-grained parallelism.

B. FPGA

The FPGA is used to accelerate the score matrix computation calculated using the data taken as input from the PC. It returns the indices for alignment as well as the score for the same. The system has been designed using Xilinx Vivado 2016.3. The RTL design was implemented using Vivado HLS and packaged as an IP to be used in the Vivado Design Suite. Furthermore, Xilinx SDK has been utilized so as to use the embedded processor.

The RTL design developed using HLS constitutes a number of techniques which make the design optimum in terms of hardware utilization and latency. The two levels of parallelism could either be implemented by pipelining the logic or by replicating the logic (*unrolling*). With experimentation, we found out that it is more suitable to unroll the loop which compares each pair (Level 1). However, due to timing constraints, the unrolling factor can be limited to only 30 (refer Improvement 1). Pipelining this led to a lot of overhead which ultimately resulted in higher latency. Furthermore, we can pipeline the modified scoring scheme which takes into account the neighbouring values to improve the throughput. As for parallelizing the calculation of elements in an anti-diagonal, we pipelined the design with an initiation interval of 2 (refer Improvement 2). This arises due to the dependency among the anti-diagonals.

Improvement 1 : Unrolling of loop for Multiple Alignment by a factor of 30

```
1 : L1 : for(i = 0; i < total; i++)
2 :     L2 : for (j = i+1; j < total; j++)
3 :         #pragma HLS UNROLL factor = 30
4 :             size[0] = all_sizes[i];
5 :             size[1] = all_sizes[j];
6 :         smith_waterman(arrays[i], arrays[j], size, res[k]);
7 :             k++;
```

Improvement 2 : Pipelining of loop for Calculation of Score Matrix

```
1 : L1 : for(i = 1; i ≤ (M+N-1); i++)
2 :         #pragma HLS PIPELINE
3 :         L2 : for(j = val; j ≤ len_diag; j++)
4 :                 #pragma HLS PIPELINE II=2
5 :                 up = score[i+1-j-1][j]-2;
6 :                 left = score[i+1-j][j-1]-2;
7 :                 if(X[j-1] == Y[i+1-j-1])
8 :                         diag = score[i+1-j-1][j-1] +2;
9 :                 else
10 :                         diag = score[i+1-j-1][j-1] -1;
11 :             score[i+1-j][j] = maximum(up,left, diag, 0);
12 :         if(i<M)
13 :                 len_diag++;
14 :         if(i>N-2)
15 :                 val = index++;
16 :         else
17 :                 val = 1;
```

The design was implemented on the Xilinx ZYNQ-7000 SoC development board, Zedboard. It has a Dual-core ARM Cortex-A9 which is the embedded processor. As mentioned in [2], the input DNA fragment pool typically has 100,000-1,000,000 arrays. A typical input array length is 15 - 100. Added to this, there are constraints on how much memory can be used at a time in the registers of the FPGA as well as the ARM Processor. Hence, the data needs to be sent in batches. The RTL design incorporates this, where the first array sent is fixed and is compared with the rest of the arrays. The hardware for each comparison has been replicated as mentioned before. We had implemented another design where-in all the arrays given as input were compared with each other. Although this led to a reduced latency per alignment, it consisted of a lot of redundant computations and hence a higher overall latency.

The synthesized Smith-Waterman IP is placed into the Programmable Logic (PL) layer of the ZedBoard. The input data for the Smith-Waterman block is fed continuously and requires communication between the host computer and the board. In cases where high speed communication protocols such as PCIe can be used, the partitioning will exhibit minimal throughput bottlenecks. The Zedboard does not have provision for PCIe communication, hence, we used UART (115200 baud) for this purpose. This however, did bottleneck the performance of the overall system. The data transfer between the various levels has been illustrated in Fig. 3. We used the AXI-ACP Protocol for PS-PL communication.

The Cypress USB-to-UART Bridge allowed connection to a host computer. The inbuilt Board support package enabled the over-riding of System I/O functions for establishing UART communication. The dataset which contained the optical label distances was read on the host computer and relayed to the board using Python. Since the optical label distances are floating point values, the data had to be unpacked and encoded to ASCII for suitable transmission over the serial port. Subsequently, the data received on the

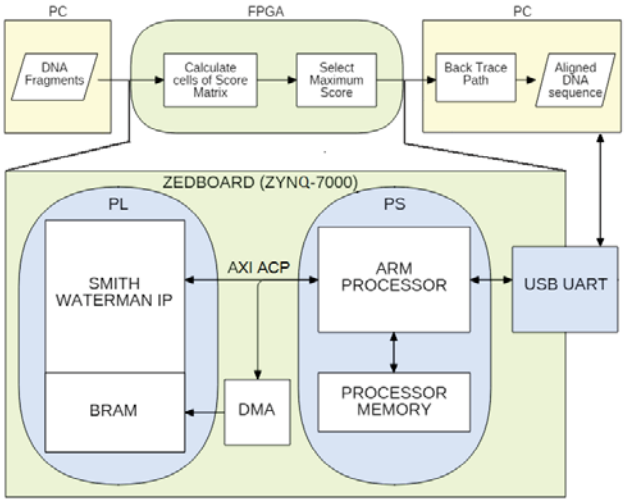

Figure 3. FPGA Implementation

board had to be decoded and repacked to floating point type. This added overhead posed as a potential bottleneck to the throughput of the entire system as a whole. Two way communication was established at 115200 baud rate, and floating point values were successfully sent to the PS Layer.

IV. Backtracking Algorithm

The result obtained from the FPGA is the best score from the pair wise evaluations. These scores are traced back to check the alignment of the pairs and hence the entire genome is built. As seen in Fig. 3, we perform backtracking on the host PC as it is not that computationally expensive and has very little scope for exploiting any sort of parallelism. The process for backtracking takes around 0.923s for aligning 375 arrays of an average length of 100 elements.

Since we send the results serially through UART, when the PC gets the required values for one pair, it gets stored in a *backtrack* array. The elements entering the arrays get stored in descending order of best score only if they have valid indices (non-negative values) for alignment. This has a maximum complexity of $O(n)$ and therefore is better than using any sorting algorithm. For our experimental dataset, the time taken for this is 0.364ms. After the transmission is complete, we apply the algorithm for backtracking and get the reconstructed genome. The algorithm for backtracking is explained in Fig. 4.

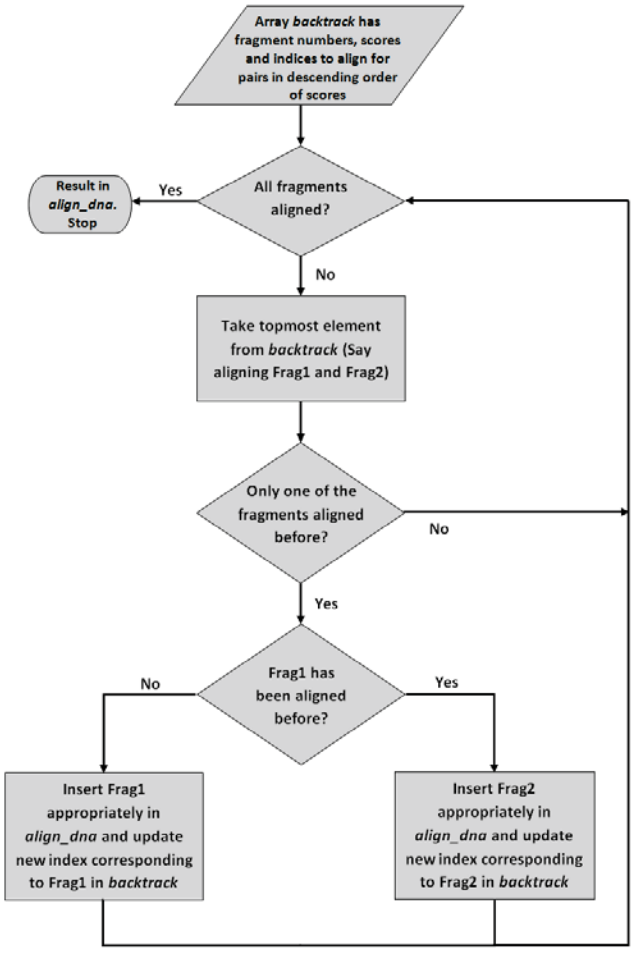

Figure 4. Algorithm for Backtracking

V. Results and Comparison

A. Local Alignment

Local alignment involves the alignment of a single pair of arrays with distances between optical labels. The modified Smith-Waterman algorithm described in section II was parallelized in C using OpenMP library. The parallelized code and the serial program were tested independently on a Quad Core Intel i7 CPU running Ubuntu ver.14.04 LTS. Initial testing involved observing the speedup obtained for local alignment with different number of threads instantiated. Keeping the input constant, the speedup for various thread counts can be seen in Fig. 5.

The highest speedup is observed for four threads which is due to the quad-core nature of the CPU used for testing. The time for four threads is approximately 2.9x times faster than sequential execution. This is in accordance the results reported by Khan *et al.* [6]. On increasing thread count, the acceleration decreases. This can be attributed to overheads of thread management on the CPU.

Once it was affirmative that four threads was the ideal count for maximum acceleration factor, the testing was carried out for different array sizes. The same alignment was also carried out on OpenMP for the modified indexing scheme explained in section II. The similar single pair alignment was also carried out on the FPGA, as explained in section III. The acceleration over sequential execution

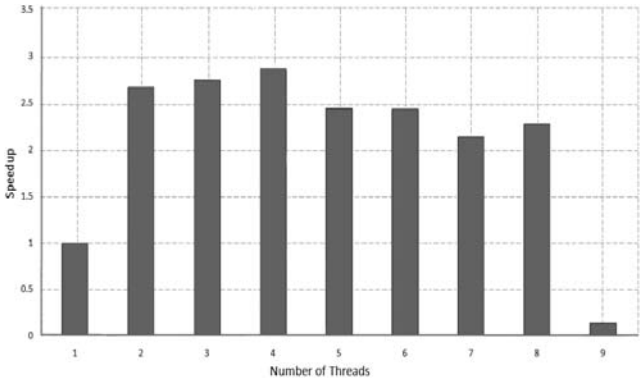

Figure 5. Speedup obtained for Single Alignment using OpenMP

was calculated for different array size The results of all the above is shown in Fig. 6. It can be noted that in the modified index scheme, even though we introduce a level of parallelism during the checking of similarity amongst the neighbouring value, it still performs worse than the scoring scheme not incorporating it. However, this is more robust and has a better accuracy during the alignment of the genome when there are a large number of contiguous repeats.

The FPGA implementation of the algorithm yields a considerably higher acceleration compared to the OpenMP implementation, with it saturating at approximately 7.02x times with increasing input size.

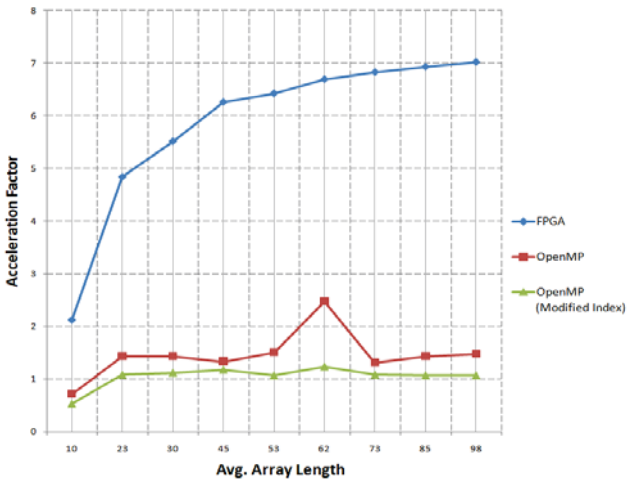

Figure 6. Performance of Single Alignment

B. Global Alignment

The local alignment when replicated for all possible comparisons among the array of fragments, results in global alignment to form the complete DNA. The multi-core CPU design was tested on the system described in the previous subsection and implemented as mentioned in section III. The dataset comprised of reads from Chromosome 10 of a human genome. There were 375 fragments of lengths varying from 12 to 97. We implemented both the cases with and without a modified index scheme. The testing was done for varying number of fragments and the acceleration factors were obtained compared to our sequential baseline design. This was run twenty times and averaged over the number of tests for higher accuracy.

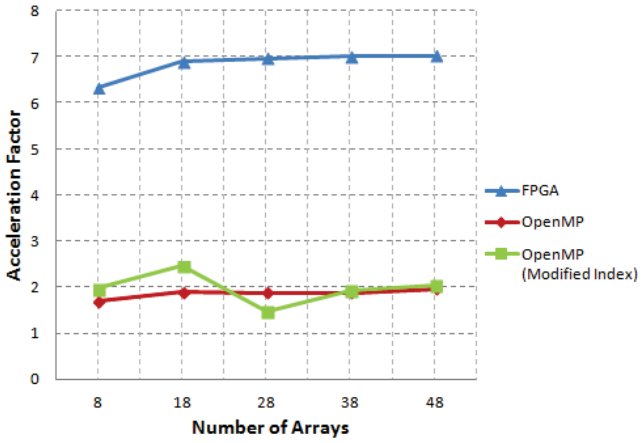

Figure 7. Analysis of Acceleration of Multiple Alignment

It can be inferred from Fig. 7 that an average acceleration of 1.86x is achieved for an OpenMP implementation. In each case, for N fragments there are $N \times (N-1)/2$ comparisons. With the addition of the modified index scheme, the performance is almost similar to the one without it. There are a few anomalies where it performs better but on an average 1.98x speedup was achieved.

The FPGA design was implemented and tested on a Zedboard development board, as mentioned in section III, with a clock frequency of 100MHz and the resource utilization details are given in Table II. The amount of data which could be pushed to the BRAM at once was limited by memory constraints on the procession system (PS) side. The testing was done using several different datasets comprising of reads from chromosome 10 of the human genome. As seen in Fig. 7, the FPGA implementation achieves a higher speedup as compared to the OpenMP design. Fig. 8 shows the trend in acceleration factor with the number of arrays. Here, the number of comparisons made is *N-1*. The acceleration factor saturates for higher number of arrays to 7.022. Lesser number of arrays renders a lot of the hardware not being utilized and hence the performance is not optimum. Since our data size is large, we keep the batch size around 300 arrays.

Fig. 9 shows the time taken to align *N-1* pairs in the software and hardware implementations. The average time taken per alignment is 0.443ms using the hardware accelerator. This time includes the time taking for computing

Table II
FPGA UTILIZATION SUMMARY

Resource	Used	Available	Utilization(%)
Slice LUTs	6968	53200	13.10%
Slice Registers	7166	106400	6.730%
BRAM	87	140	62.14%
DSP48E1	1	220	0.45%
BUFG	1	32	3.13%

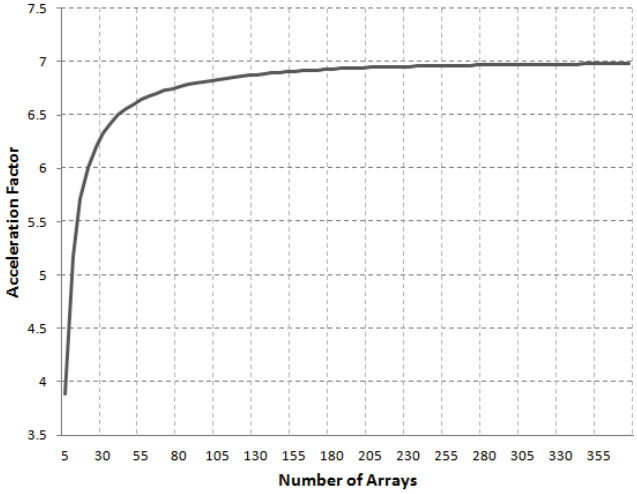

Figure 8. Trend in Acceleration Factor with Dataset Size

the score matrix, best score, alignment indices and the time taken to transfer data between the PS and PL using AXI ACP. When the modified index calculation is incorporated in the design, not only does the utilization of resources on the FPGA increase, the overall acceleration achieved also reduces slightly to 6.78x. However, as mentioned before, we employ this modification only in cases where there are large number of continuous repeats and we want a better accuracy.

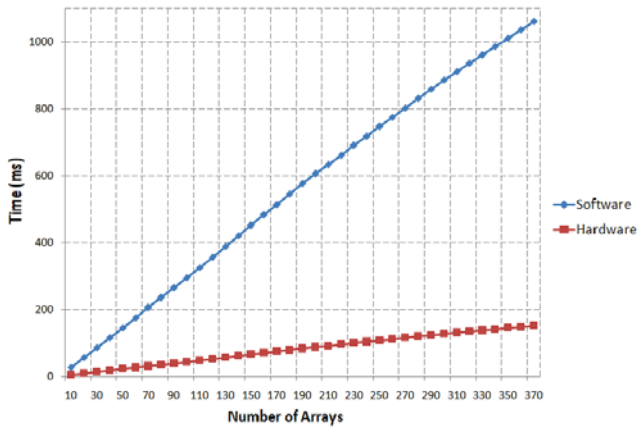

Figure 9. Comparison of Software and Hardware Implementation

UART was the major bottleneck in communication from the PC to the board, which resulted in a transmission time of 9.775s to transmit the entire data. Furthermore, backtracking to align the entire DNA happens quickly in 0.923s. Hence, we were able to successfully align the reads from chromosome 10 of the human genome and reconstruct it without any error.

VI. CONCLUSION

We have proposed a novel algorithm based on the Smith-Waterman for the large-scale alignment of optically labeled genomes using a modified version of the Smith-Waterman algorithm in this paper. This algorithm is specific to for the alignment of human genome. Parallelism from this algorithm has been exploited for the multi-core CPU and FPGA design. The design resulted in a 1.98x speedup in the multi-core CPU design with both coarse and fine-grain parallelism in OpenMP. A complete end to end design for DNA alignment was implemented on a Zedboard development board. Using techniques like pipelining and unrolling resulted in a 7.022x speedup for the alignment process. Furthermore, the modified index calculation scheme improves the accuracy of alignment. We reckon that a higher acceleration factor and better resource utilization can be achieved by developing a custom RTL design rather than using HLS.

REFERENCES

[1] A. R. Hastie, L. Dong, A. Smith, J. Finklestein, E. T. Lam, N. Huo, H. Cao, P.-Y. Kwok, K. R. Deal, J. Dvorak, M.-C. Luo, Y. Gu, and M. Xiao, "Rapid genome mapping in nanochannel arrays for highly complete and accurate de novo sequence assembly of the complex aegilops tauschii genome," *PLOS ONE*, vol. 8, pp. 1–10, 02 2013.

[2] P. Meng, M. Jacobsen, M. Kimura, V. Dergachev, T. Anantharaman, M. Requa, and R. Kastner, "Hardware accelerated novel optical de novo assembly for large-scale genomes," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, Sept 2014.

[3] A. Valouev, *Shotgun Optical Mapping: A Comprehensive Statistical and Computational Analysis*. PhD thesis, Los Angeles, CA, USA, 2006. AAI3238339.

[4] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195 – 197, 1981.

[5] H.-Y. Liao, M.-L. Yin, and Y. Cheng, "A parallel implementation of the Smith-Waterman algorithm for massive sequences searching," in *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2, pp. 2817–2820, Sept 2004.

[6] A. Khan, L. Hassan, and S. Ullah, "Open MP-based parallel and scalable Genetic Sequence alignment," *Journal of Engineering and Applied Sciences (JEAS), University of Engineering and Technology, Peshawar*, vol. 34, no. 2, 2015.