Accelerating Bitap on FPGA for Genomic Sequences

Gayatri Madduri
IIIT, Hyderabad
Hyderabad, India
gayatri.madduri@students.iiit.ac.in

P Sahithi Reddy
IIIT, Hyderabad
Hyderabad, India
p.sahithi@research.iiit.ac.in

***Abstract*—The Approximate String Matching(ASM) problem is a crucial and computationally intensive step in genome sequencing. This problem is often solved using Dynamic Programming(DP) based methods,which have a quadratic time and space complexity. Alternatively, Bitap algorithm(of significantly lower time and space complexities) based on bitvectors and bitwise manipulations can be used to solve ASM. Our project aims to accelerate Bitap-based solution on an FPGA.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Genomic Sequencing is a comprehensive method for analysing entire genomes. DNA sequencing plays pivotal role in enabling many medical and scientific advancements in personalized medicine, evolution theory and forsenics. Human DNA is generally 3 billion base pairs long. Modern sequencing machines generate large amounts of genomic data at low costs, but they can't generate in one piece.These machines generate smaller random fragments of DNA called *reads*. Hence these reads should be put together to analyze the DNA. This process of mapping reads using a reference genome using computational process is called *read mapping*. During read mapping each read is aligned to one or more possible locations within the reference genome, and the matches and differences (i.e., distance) between the read and the reference genome segment at that location are found.

Modern sequencing machines generate two types of reads *short reads* and *long reads*. *Short reads* are generally few hundred base pairs long and have error rate upto 0.1%. Compared to billion base pairs in DNA, these reads are short and pose non-deterministic mapping and computational challenges.On the other hand long reads are generally thousand to million of base-pairs long and have error rate upto 10-15%. For both short reads and long reads there are sequencing errors and variations from reference genome due to genetic mutations and variations.These errors and differences take the form of base insertions, deletions, and/or substitutions.

Hence during various steps of read mapping we need to account these errors, therefore we need to perform approximate string matching (ASM) between reads and candidate mapping locations in reference genome.The measure of similarity between two strings is given by *edit distance*.Edit distance is the number of substitute, insert or delete operations that are required to transform one string to the other.The ASM algorithms are generally bottleneck of read-mapping and are based on compute heavy dynamic programming.In this paper we discuss one such algorithm called bitap algorithm which is based on simple bit-wise operations to calculate *edit distance*.

II. Related Work

[1] discusses the Approximate String Matching(ASM) problem, which is crucial to align a read to the correct position in the reference genome. It provides an acceleration framework based on Bitap algorithm.

Bitap algorithm aims to find all possible starting locations where the read aligns with the reference genome. This involves using k+1 status bit vectors. The i^{th} status bit vector represents the alignment status with up to i errors.

Bitap algorithm involves iterating over every character in the reference genome and updating the status bit vectors. At every text iteration, we update all the $k+1$ status bit vectors. For brevity, we skip the details of the algorithm. It is important to note that at text iteration i, computing the status bit vector $R[j]$ requires $R[j-1]$ from text iterations i and $i-1$ and $R[j]$ from text iteration i.

[1] proposes Genasm-DC, which makes necessary changes to Bitap algorithm such as removal of loop dependencies and text-level parallelism to make it suitable for parallel computing. A CPU based implementation and an ASIC to accelerate the algorithm are provided.

However, there is no existing method to accelerate these algorithms on FPGA. FPGAs are viable here due to their cost efficiency and parallel computing capacities. Our work aims to accelerate he algorithm on an FPGA. This can significantly reduce the time taken for read alignment and boost up the process of genome sequencing analysis.

III. Experimental Work

1) Our plan to tackle the loop dependencies and data transfer overhead while running algorithm on an FPGA such that it gives better performance compared to CPUs.
2) We will start with referring to the algorithm and code for CPU parallelisation provided in the github repository: https://github.com/CMU-SAFARI/GenASM.git
3) We extract datasets from [2]. The input datasets to [2] are found at : https://www.ebi.ac.uk/ena/data/view/ERR240727

https://www.ebi.ac.uk/ena/data/view/SRR826460
https://www.ebi.ac.uk/ena/data/view/SRR826471

IV. Method

The algorithm solves for the minimum edit distance between a query and a text with a maximum of k errors. The Algorithm begins with a pre-processing step. In this step, pattern masks are generated for the query string. Figure 1 illustrates this step. The next step is to update the status bitvectors. These status

	C	T	G	A
PM(A)	1	1	1	0
PM(G)	1	1	0	1
PM(C)	0	1	1	1
PM(T)	1	0	1	1

Figure 1. Pattern masks for the query

bitvectors,each of the same length as the query, are k in number where k is the edit distance threshold. We maintain two sets of status bitvectors $R[d]$ and oldR$[d]$, all of which are initialised to $11\cdots1$.
The Algorithm then has text_length iterations. In each of these iterations i, $R[d]$ and oldR$[d]$ is computed for the text character i. The lines 13 through 19 illustrate this step in the algorithm in Figure 2.

Algorithm 1 Bitap Algorithm

Inputs: text (reference), pattern (query), k (edit distance threshold)
Outputs: startLoc (matching location), editDist (minimum edit distance)

```
1: n ← length of reference text
2: m ← length of query pattern
3: procedure PRE-PROCESSING
4:     PM ←generatePatternBitmaskACGT(pattern)   ▷ pre-process the pattern
5:     for d in 0:k  do
6:     |   R[d] ← 111..111                        ▷ initialize R bitvectors to 1s
7: procedure EDIT DISTANCE CALCULATION
8:     for i in (n-1):-1:0 do                    ▷ iterate over each text character
9:     |   curChar ← text[i]
10:    |   for d in 0:k  do
11:    |   |   oldR[d] ← R[d]      ▷ copy previous iterations' bitvectors as oldR
12:    |   curPM ← PM[curChar]                   ▷ retrieve the pattern bitmask
13:    |   R[0] ← (oldR[0]<<1) | curPM           ▷ status bitvector for exact match
14:    |   for d in 1:k  do                      ▷ iterate over each edit distance
15:    |   |   deletion (D) ← oldR[d-1]
16:    |   |   substitution (S) ← (oldR[d-1]<<1)
17:    |   |   insertion (I) ← (R[d-1]<<1)
18:    |   |   match (M) ← (oldR[d]<<1) | curPM
19:    |   |   R[d] ← D & S & I & M              ▷ status bitvector for d errors
20:    |   if MSB of R[d] == 0, where 0 ≤ d ≤ k  ▷ check if MSB is 0
21:    |   |   startLoc ← i                      ▷ matching location
22:    |   |   editDist ← d                      ▷ found minimum edit distance
```

Figure 2. Bitap Algorithm

V. Acceleration Methods Used

Because of the data dependencies illustrated in the figure below, we cannot directly parallelize either of the two for loops present in the Bitap Algorithm.
However, we can observe here that in a given cycle, there are no dependencies. Thus, Text$[4]R_0$ and Text$[3]R_0$ do

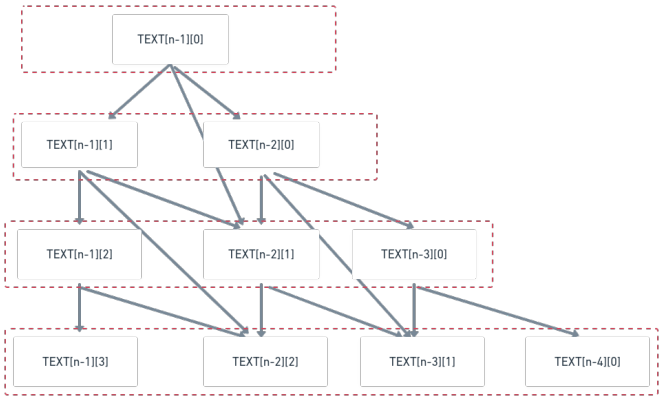

Figure 3. Dependencies in the algorithm

not have dependencies between them and can be computed independently. For this, the algorithm has to be transformed as:

Algorithm 2 Parallelised Bitap's algorithm

```
for j in 0:k+n do
  for i in min(j,k):0:-1 do
    curchar ← text[i]
    curpm ← PM(curchar)r
    if i==0 then
      R[n − j + i][i] ← R[n − j + i + 1][i − 1]
    else
      deletion ← R[n − j + i + 1][i − 1]
      insertion ← R[n − j + i][i − 1] << 1
      substitution ← R[n − j + i + 1][i − 1] << 1
      matching ← (R[n − j + i + 1][i − 1])|curpm
      R[n − j + i][i] ← D&I&S&M
```

Figure 4. Changes in Bitap for parallelisation

We further use Binding methods to store and flush the output on kernel side.We are binding input and all the R values onto BRAM of FPGA using RAM_2P, which allows read on one port and both read and write on another port.For larger values of R when limit of BRAM exceeds, we can replace few values, as we have only two level dependency, we do not need values prior to the 2 cycles from the current one. Moreover we are using multiple axi ports for transferring memory bundles.

VI. Results

The following are some results observed:

VII. Roofline Analysis

We start by computing the computational roof. We use the number of DSPs and the clock frequency to calculate the computational roof as $6800 \times 600 = 4080$ GFlops.
Let query size be represented by Q and text size be represented by T. We know that the memory roof is given by the total memory bandwidth i.e, $16GB$.
We calculate the *Memory traffic* is given by $Q \times 32 + T$.
To calculate operations performed per data transfer, we

```
<terminated> (exit value: 0) SystemDebugger_Bitap_system_Bitap [OpenCL] /home/ec2-user/workspace/Bitap/Emulation-HW/Bitap (12/9/21, 6:03 PM)
[Console output redirected to file:/home/ec2-user/workspace/Bitap/Emulation-HW/SystemDebugger_Bitap_system_Bitap.launch.log]
-----Accelerating bitap algorithm----------
Loading/home/ec2-user/workspace/Bitap_system/Emulation-HW/binary_container_1.xclbinto program
INFO: [HW-EMU 01] Hardware emulation runs simulation underneath. Using a large data set will result in long simulation times.
configuring penguin scheduler mode
scheduler config ert(0), dataflow(1), slots(16), cudma(1), cuisr(0), cdma(0), cus(1)
length of pattern is 4length of text is 7
CPU result
success, edit distance is0INFO::[ Vitis-EM 22 ] [Time elapsed: 0 minute(s) 42 seconds, Emulation time: 0.152899 ms]
Data transfer between kernel(s) and global memory(s)
wide_vadd_1:m_axi_gmem-DDR[1]          RD = 0.016 KB               WR = 0.000 KB
wide_vadd_1:m_axi_gmem1-DDR[1]          RD = 0.000 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem2-DDR[1]          RD = 0.004 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem3-DDR[1]          RD = 0.000 KB              WR = 0.004 KB

INFO: [HW-EMU 06-0] Waiting for the simulator process to exit
INFO: [HW-EMU 06-1] All the simulator processes exited successfully
```

Figure 5. Outputs

```
Version:    v++ v2021.1 (64-bit)
Build:      SW Build 3246112 on 2021-06-09-14:19:56
Copyright:  Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
Created:    Thu Dec  9 17:48:51 2021
================================================================================

--------------------------------------------------------------------------------
Design Name:              wide_vadd
Target Device:            xilinx:aws-vu9p-f1:shell-v04261818:201920.2
Target Clock:             250.000000MHz
Total number of kernels: 1
--------------------------------------------------------------------------------

Kernel Summary
Kernel Name  Type  Target            OpenCL Library  Compute Units
-----------  ----  ----------------  --------------  -------------
wide_vadd    c     fpga0:OCL_REGION_0  wide_vadd       1


--------------------------------------------------------------------------------
OpenCL Binary:    wide_vadd
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit  Kernel Name  Module Name  Target Frequency  Estimated Frequency
------------  ----------  -----------  ----------------  -------------------
wide_vadd_1   wide_vadd   wide_vadd    250               inf

Latency Information
Compute Unit  Kernel Name  Module Name  Start Interval  Best (cycles)  Avg (cycles)  Worst (cycles)  Best (absolute)  Avg (absolute)  Worst (absolute)
------------  ----------  -----------  --------------  -------------  ------------  --------------  ---------------  --------------  ----------------
wide_vadd_1   wide_vadd   wide_vadd    1               0              0             0               0 ns             0 ns            0 ns

Area Information
Compute Unit  Kernel Name  Module Name  FF   LUT  DSP  BRAM  URAM
------------  ----------  -----------  ---  ---  ---  ----  ----
wide_vadd_1   wide_vadd   wide_vadd    530  944  0    0     0
--------------------------------------------------------------------------------
```

Figure 6. Kernel estimate

observe that total number of operations carried out is $(T \cdot K \cdot 4)$.
Considering that T and Q can be maximum $100bp$ long and K can be a maximum of 10, Operation intensity is given by $40/32 = 5/4$ in the extreme case. Thus, the operational intensity reduces to $\frac{T \cdot K \cdot 4}{T+Q \cdot 32}$. We also observe that throughput$= 600/$(outputs per cycle produced)
Thus, the output Memory Bandwidth required is also equal to $600/$(outputs per cycle produced).
The input Memory Bandwidth requirement is given by $600 \cdot 5 = 3000$

1) If our current input memory bandwidth requirement is less than 16GB/s, it's compute-bound. That means we need to increase the no. of computations we do.
2) Else it would be memory bound. In that, we can increase PCIe bandwidth.

In the following graph, the red dot roughly represents the current design.

OI at which peak is achieved would be = 4080/16 = 245

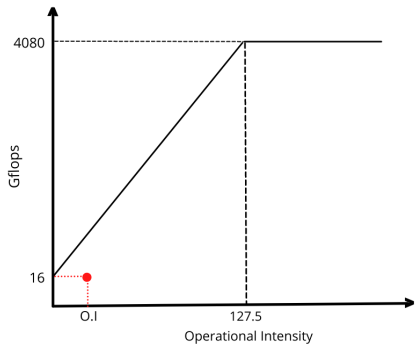

Figure 7. Dependencies in the algorithm

REFERENCES

[1] F. Zhu, Z. Liang, X. Jia, L. Zhang, and Y. Yu, "A benchmark for edge-preserving image smoothing," *IEEE Transactions on Image Processing*, vol. 28, no. 7, pp. 3556–3570, 2019.

[2] H. X. et al., "Accelerating read mapping with fasthash," *BMC Genomics, 14(Suppl 1):S13, 21 January 2013)*, 2013.