Supplementary Materials: FPGA-based Accelerator for Adaptive Banded Event Alignment in Nanopore Sequencing Data Analysis

Yilin Feng†, Zheyu Li†, Gulsum Gudukbay Akbulut, Vijaykrishnan Narayanan, Mahmut Taylan Kandemir, Chita R. Das

Department of Computer Science and Engineering, The Pennsylvania State University, 201 Old Main, University Park, 16802, PA, USA.

*Corresponding author(s). E-mail(s): ypf5071@psu.edu;
†These authors contributed equally to this work.

1 Background of Genome Sequencing

1.1 DNA Sequencing

Firstly, some concepts of DNA sequencing are given as follows. A genome contains all genetic information of an organism or a cell, consisting of DNA or RNA nucleotide sequences. DNA sequencing is the process of determining the order of bases (AGCT) of DNA. Most modern sequencing technologies fragment the genome into millions of molecules and each fragment produces a DNA sequence referred to as a read. The term k-mer of a sequence refers to all substrings of length k. For example, 'AGAT' has three 2-mer which are 'AG', 'GA', and 'AT'. Sequence alignment is a foundational technique to identify the similarities and differences among multiple DNA, RNA, or protein sequences. It involves finding the optimal arrangement of sequences by matching similar regions and identifying insertions, deletions, or substitutions.

Over the years, generations of sequencing technologies have evolved, offering increased accuracy and throughput. The advent of DNA sequencing can be traced back to 1977 with the introduction of Sanger sequencing [9], marking the inception of First Generation Sequencing Technology. Since 2005, Next Generation Sequencing has emerged, capable of generating millions to billions of short reads (50 to 400 bases) in a single run. In the 2010s, Third Generation Sequencing began to take shape.

These advanced sequencing technologies can generate substantially longer reads, up to 30,000 bases, surpassing the capabilities of Next Generation Sequencing. The increased length of these reads provides richer information, leading to significant advancements in genome assembly, detection of rare variants, and numerous other applications in biology and medicine.

1.2 Nanopore Sequencing and Analysis

Nanopore sequencing [2, 6] is a cutting-edge third-generation sequencing technology, capable of real-time analysis of long DNA or RNA fragments. It operates by passing DNA or RNA molecules through a nanopore—a tiny hole embedded in a membrane. As the molecules move through the nanopore, they disrupt an ionic current, generating a signal that is used to identify the specific sequence of nucleotides. Oxford Nanopore Technology(ONT)'s PromethION 48, a high-throughput sequencing device, can produce up to 14 Tb of data over 72 hours at a rate of 420 bases per second. Consequently, the computational analysis of ONT-generated data poses a significant challenge [8].

The raw signals are converted into nucleotide strings by a base-calling process. However, this method typically exhibits a higher error rate, ranging from 5% to 15%, compared to other sequencing technologies like Illumina sequencing [1, 3]. Consequently, additional downstream processing steps are required to improve the accuracy and quality of the sequencing data. Firstly, the base-called read is aligned to a reference genome, such as the human genome, to identify similarities. Subsequently, the polishing process revisits raw signals, conducting a signal-level alignment using the alignment results from the previous step. This facilitates error correction and enables the detection of modified bases.

DNA methylation, which involves the addition of a methyl group (CH3) to cytosine or adenine nucleotide bases, plays a crucial role in epigenetics and medicine research. Methylation detection represents a standard Nanopore data analysis workflow employed to identify methylation patterns. In the context of Oxford Nanopore sequencing data analysis, methylation detection is typically conducted using a four-stage process, as depicted in Figure 1, and implemented in *Nanopolish* [10]. The analysis workflow starts with loading the raw signal data and base-called reads in the first stage. The second stage involves time series segmentation to detect events, which are characterized by abrupt changes in signal levels. These segments, identified by parameters such as mean, standard deviation, and duration, are referred to as events. Subsequently, these events are aligned to k-mers of base-called reads to derive accurate sequence annotations, utilizing the Adaptive Banded Event Alignment (ABEA) algorithm in the third stage. The final stage employs a hidden Markov model (HMM) to determine the methylation states of bases [5]. Table 1 shows the CPU execution time breakdown of *Nanopolish* on an example dataset. Notably, the ABEA algorithm accounts for approximately 70% of the total processing time.

2 FPGA Programming

Field-Programmable Gate Arrays(FPGAs) are a type of device that can be programmed to any specific digital design. An FPGA device consists of a sea of

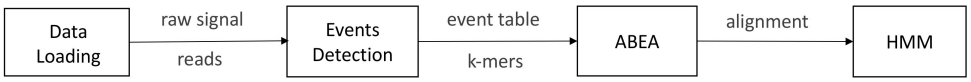

Fig. 1 Methylation detection workflow

Table 1 *Nanopolish* processing time in second

Total time	Event Detection	ABEA	Calibration	HMM
74	8	51	2	14

programmable logic blocks including Look-Up-Tables(LUTs), block RAMs(BRAMs), Digital Signal Processing blocks(DSPs), and programmable interconnects, such that FPGA devices exhibit massive inherent parallelism and large high-bandwidth low-latency on-chip memories. Off-chip memory, such as Dynamic Random Access Memory (DRAM) and HBM (High Bandwidth Memory) are also commonly attached to FPGA devices to interface large amounts of data which does not fit in on-chip BRAMs. Hardware Descriptive Languages(HDL) such as VHDL or Verilog are traditionally used to design, simulate, synthesize, and implement desired functions to FPGA devices. Recently, C/C++-based High-Level Synthesis(HLS) tools such as Vivado HLS [7] have shown increasing adoption for numerous reasons: ease of programming, rapid simulation/debugging ability, and comparable synthesizing quality to HDL. The fine-grained programmability and massive parallelism of FPGA devices make it an ideal platform to accelerate computation/data-intensive bioinformatics workloads[4].

3 Post Stage

Algorithm 1 illustrates the Post Stage in our accelerator, which performs backtracking for N reads processed by Compute Stage. Lines 1-5 initialize essential variables *event_idx*, *kmer_idx* and *skip*. *skip* is a boolean flag used to decide whether to skip the previous band during backtracking. The core backtracking loop from line 6 to line 39 deal with N reads in a pipelined and parallelized fashion using `pragma pipeline` and `pragma unroll` directives. `pragma pipeline` allows the operations of the loop to be executed concurrently. `pragma unroll` unrolls the loop by a factor of n, creating n copies of the loop and reducing the number of iterations by a factor of 1/n. A factor of 2 is used in our implementation, thus, 2 bands from 2 reads are processed in parallel each cycle. During backtracking, the algorithm reads the streams of *trace_table* until reaching the position where max score is obtained. It then processes bands in reverse order for each read in lines 12-13. There are tree types of flags ($FROM\_D$, $FROM\_U$ and $FROM\_L$) correspond to score calculation dependence on cell from the diagonal, the up and the left in the *trace_table* and the algorithm updates *kmer_id* and *event_id* according to recorded flags. If the flag indicates a diagonal dependency, the algorithm sets *skip* to true to skip the previous band. The path of the backtracking is recorded as pairs of (*kmer_id*,*event_id*) and is written to *aligned_pairs* as output.

Algorithm 1 Post Stage

Input: *trace_table, max_event_idx, n_event, n_kmer, events, model*
Output: *n_pairs,aligned_pairs*

```
1: for i ⇐ 0 to N do
2:     event_idx[i] ⇐ n_event[i] -1          ▷ event_idx is initialized as the last event
3:     kmer_idx[i] ⇐ n_kmer[i] -1            ▷ kmer_idx is initialized as the last k-mer
4:     skip[i] ⇐ false                       ▷ flag to skip the previous band
5: end for
6: for i ⇐ max_band-1 to 0 do
7:     #pragma HLS pipeline
8:     #pragma HLS UNROLL FACTOR=2           ▷ The loop is unrolled by 2
9:     for j ⇐ 0 to N  do
10:        kmer_id ⇐ kmer_idx[j]
11:        event_id ⇐ event_idx[j]
12:        read stream until max_event_idx
13:        trace_table.read(temp_trace)
14:        if skip[j] == false then
15:            offset ⇐ get_offset(event_idx)
16:            from ⇐ temp_trace[offset]
17:            if from == FROM_D then
18:                events[j].read(cur_event)
19:                model[j].read(cur_model)
20:                event_idx[j] ⇐ event_idx[j]-1
21:                kmer_idx[j] ⇐ kmer_idx[j]-1
22:                skip[j] ⇐ true
23:            else if from == FROM_U then
24:                events[j].read(cur_event)
25:                event_idx[j] ⇐ event_idx[j]-1
26:                skip[j] ⇐ false
27:            else
28:                model[j].read(cur_model)
29:                kmer_idx[j] ⇐ kmer_idx[j]-1
30:                skip[j] ⇐ false
31:            end if
32:        else
33:            kmer_id ⇐ kmer_id - 1
34:            event_id ⇐ event_id -1
35:            skip[j] ⇐ false
36:        end if
37:        aligned_pairs[j].write(kmer_id, event_id)
38:    end for
39: end for
```

4 Bucket Scheduling

Algorithm 2 is the pseudocode for our bucket scheduling approach, which partitions reads into buckets based on length and assigns each bucket to a specific CU for processing. The key intermediate arrays, *reads_num*, *max_event_length*, *max_read_length*, *bucket_reads* and *ultra_long_reads* are declared in line 1. These arrays store the number of reads per bucket, the maximum event length (number of events) and the maximum read length for each bucket, the reads assigned to each bucket, and a list of ultra-long reads, respectively. The essential parameters, *RANGE* and *BUCKET_SIZE* define the length range for each bucket and the number of reads per bucket. For instance, if *RANGE* is set to 10,000, lengths interval for buckets will be [0, 10,000), [10,000, 20,000), [20,000, 30,000), etc. Lines 4-8 collects ultra-long reads into a list and call the Ultra Long CU for alignment in a separate thread. The main loop (lines 10-22) then groups reads into buckets based on their length (sum of event length and read length). Line 12 decides the bucket index *k*, by dividing the length by *RANGE*, and line 13 appends the read to the corresponding bucket list. Lines 14-15 update *max_event_length* and *max_read_length* for the bucket, which are used to determine data intervals and processing time in one bucket. Once a bucket is full, it is transferred to FPGA DRAM for alignment by the Pipeline CUs(lines 17-20). Finally, unfilled buckets are processed in pipeline too, as depicted in lines 23-27.

References

[1] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, et al. Accurate whole human genome sequencing using reversible terminator chemistry. *nature*, 456(7218):53–59, 2008.

[2] D. Branton, D. W. Deamer, A. Marziali, H. Bayley, S. A. Benner, T. Butler, M. Di Ventra, S. Garaj, A. Hibbs, X. Huang, et al. The potential and challenges of nanopore sequencing. *Nature biotechnology*, 26(10):1146–1153, 2008.

[3] M. J. Chaisson and G. Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC bioinformatics*, 13:1–18, 2012.

[4] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang. Fpga hls today: Successes, challenges, and opportunities. *ACM Trans. Reconfigurable Technol. Syst.*, 15(4), aug 2022.

[5] Y. Feng, G. Gudukbay Akbulut, X. Tang, J. R. Gunasekaran, A. Rahman, P. Medvedev, and M. Kandemir. GPU-accelerated and pipelined methylation calling. *Bioinformatics Advances*, 2(1):vbac088, Nov 2022.

[6] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338–345, 2018.

[7] AMD. Vivado Design Suite User Guide: High-Level Synthesis. https://docs.amd.com/v/u/en-US/ug902-vivado-high-level-synthesis, 2021.

Algorithm 2 Bucket Scheduling

Declare array: reads_num[], max_event_length[], max_read_length[], bucket_reads[][], ultra_long_reads[]
RANGE $\Leftarrow$ *10000*
BUCKET_SIZE $\Leftarrow$ *128*
for *i* $\Leftarrow$ *0 to N_Reads* **do**
    if *reads[i].event_length* $\geq$ *120000* **or** *reads[i].read_length* $\geq$ *60000* **then**
        ultra_long_reads.append(reads[i])
    end if
end for
align_ultra_long_async(ultra_long_reads[]) ▷ launch a thread to call Ultra Long CU to process ultra-long reads
for *i* $\Leftarrow$ *0 to N_Reads* **do**
    if *reads[i].event_length* $\leq$ *120000* **and** *reads[i].read_length* $\leq$ *60000* **then**
        k $\Leftarrow$ *(reads[i].event_length + reads[i].read_length)/RANGE*
        bucket_reads[k].append(reads[i])
        max_event_length[k] $\Leftarrow$ *max(max_event_length[k], reads[i].event_length)*
        max_read_length[k] $\Leftarrow$ *max(max_read_length[k], reads[i].read_length)*
        reads_num[k]++
        if *reads_num[k] == BUCKET_SIZE* **then** ▷ call Pipeline CU
            align_pipeline(bucket_reads[k],max_event_length[k],max_read_length[k])
            reads_num[k], max_event_length[k], max_read_length[k] $\Leftarrow$ *0*
        end if
    end if
end for
for *k* $\Leftarrow$ *0 to N_Buckets* **do** ▷ process remaining unfilled buckets
    if *bucket_reads[k] is not NULL* **then**
        align_pipeline(bucket_reads[k],max_event_length[k],max_read_length[k])
    end if
end for
align_ultra_long_async_join() ▷ wait ultra long reads processing finishes

[8] Oxford Nanopore Technologies. Oxford Nanopore Technologies. https://nanoporetech.com, 2023.

[9] F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, 1977.

[10] J. T. Simpson, R. E. Workman, P. C. Zuzarte, M. David, L. J. Dursi, and W. Timp. Detecting dna cytosine methylation using nanopore sequencing. *Nature Methods*, 14(4):407–410, Apr 2017.