A Methodology for Accelerating Variant Calling on GPU

Beatrice Branchini[1, 2], Alberto Zeni[2], Marco D. Santambrogio[2]
[1]Pacific Northwest National Laboratory, Richland (WA, USA)
[2]Politecnico di Milano, Milan (Italy)

1 INTRODUCTION

The medical domain is shifting toward a personalized approach to healthcare delivery. To achieve such a goal, the community requires new high-performance tools to find genetic mutations in the patient's genome. In this context, the Broad Institute's Genome Analysis ToolKit HaplotypeCaller represents one of the most used tools to highlight such variants. The core of this tool is the Pair Hidden Markov Model (PairHMM) algorithm, which, however, embodies a considerable bottleneck for the entire application. Offloading the execution of this task to hardware accelerators represents a valuable solution. Nevertheless, State-of-the-Art designs lack flexibility for managing the significant variability of the size of the inputs the tool has to process [1]. They exploit specific features within datasets, making them only applicable to specific scenarios. Consequently, they are intrinsically defective, and their applicability in standard clinical practice is considerably limited. For these reasons, this work presents a novel high-performance GPU accelerator for the PairHMM algorithm that overcomes the fixed-length limitation of State-of-the-Art designs. To summarize, this work provides the following contributions:

- a *dynamic memory swap* methodology that allows tailoring memory resources to increase the throughput while preserving the application's usability;
- a high-performance GPU implementation of the PairHMM algorithm, which outperforms the fastest State-of-the-Art hardware-accelerated solution up to 1.6× and achieving an 8154× speedup over the software baseline.

2 PAIRHMM ALGORITHM

The PairHMM relies on the computation of three Dynamic Programming (DP) matrices (M, X, Y), which allows calculating the comprehensive alignment probability. Algorithm 1 describes the DP-based formulation for the PairHMM given a *read R* and an *haplotype H*. The recursive equations to compute M, X, and Y are:

$$\begin{cases} M(i,j) = p_j * \{mm_j * M(i-1,j-1) + gm_j * [X(i-1,j-1) + Y(i-1,j-1)]\} \\ X(i,j) = xx_j * X(i,j-1) + mx_j * M(i,j-1) \\ Y(i,j) = yy_j * Y(i-1,j) + mx_j * M(i-1,j) \end{cases} \tag{1}$$

In these equations, *mm, mx, my, gm, xx,* and *yy* are *transition probabilities* and represent the probability of a state transition. Their value depends on the sequences quality scores, supplied by the sequencing machinery. p_j is the *prior probability (p)* and depends on the base quality score, Q_b, supplied by the biological sequencing process. It assumes different values:

$$p = \begin{cases} \epsilon(Q_b)/3 & \text{if mismatching bases} \\ 1 - \epsilon(Q_b) & \text{if matching bases} \end{cases} \tag{2}$$

where $\epsilon(Q_b)$ is the error rate calculated from the base quality in Phred scale [3].

Algorithm 1 PairHMM algorithm

```
1: Initializations: $M_{0,j} = X_{0,j} = 0$ and $Y_{0,j} = 2^{1020}/len(H)$ for
   $1 \leq j \leq len(H)$
2: for $1 \leq i \leq len(R)$ do
3:     for $1 \leq j \leq len(H)$ do
4:         Calculate $M_{ij}$ with Equation (1)
5:         Calculate $X_{ij}$ with Equation (1)
6:         Calculate $Y_{ij}$ with Equation (1)
7:     end for
8: end for
9: Total likelihood: $P(R|H) = \sum_j \left(M_{R,j} + I_{R,j}\right)$
```

3 IMPLEMENTATION

The computation of the sequence similarity with the PairHMM algorithm can be parallelized by exposing a wavefront-based computational pattern, following the same approach as in the Smith-Waterman [9] and Needleman-Wunsch [6] algorithms. The wavefront pattern exploits the lack of dependencies among cells in the same anti-diagonal in the DP matrix, enabling us to update them in parallel. Hence, we map each thread to an element of the anti-diagonal to parallelize the computation. To ensure the computation of anti-diagonals of any length, we split each anti-diagonal into segments, whose width is equal to the number of threads within a block. We also leverage the limited number of anti-diagonals involved at each iteration to reduce the kernel memory footprint, storing only the three needed for the specific iteration. We allocate the anti-diagonals in shared or global memory, depending on the resource required, determined according to the dynamic memory swap (Section 3.1). We then focused on exposing another layer of parallelism to improve device utilization. Therefore, we perform multiple alignments in parallel on the device by mapping each pair of sequences to a GPU block. In this case, the number of alignments determines the number of blocks spawned on the device.

3.1 Dynamic Memory Swap

To maximize the flexibility and the performance of our solution, we propose a new methodology for accelerating the PairHMM execution. The *dynamic memory swap* relies on the observation that the memory required by the computed anti-diagonals depends only on the length of the sequences involved in the alignment. Therefore, pre-computing the required resources for each alignment allows for fine-grained tailoring of the allocated memory at runtime. We exploit this knowledge by dynamically allocating GPU shared memory when invoking the kernel, while we also allocate GPU global memory for the longer alignments that exceed the maximum amount of allocatable shared memory. This enables us to support any alignment length and to exploit the most performant GPU memory for every workload.

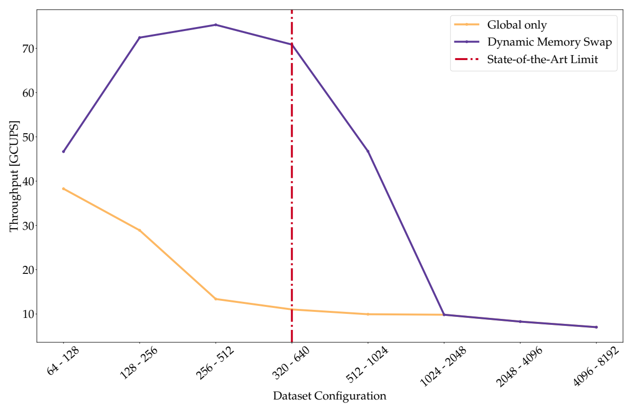

Figure 1: Throughput with (violet) and without (yellow) the dynamic memory swap on the NVIDIA V100; red line refers to the maximum supported length in State-of-the-Art designs

4 RESULTS

4.1 Experimental Setup

We tested our solution on an NVIDIA V100 connected to a dual-socket Intel Xeon Platinum 8167M CPU with 768 GB of RAM. In all evaluations, we consider the *kernel time*, measured with NVIDIA Nsight Systems. In addition, we compute the throughput in Giga Cell Updates Per Second (GCUPS) as follows:

$$GCUPS = \frac{\sum_i (rlen_i \times hlen_i)}{t \times 10^9} \quad (3)$$

where *rlen* and *hlen* are the *i*-th read and haplotype lengths, respectively, and *t* is the kernel time in seconds. For the first evaluation, we generated eight synthetic datasets comprising one million alignments with different sequence lengths. The haplotypes span from 128 to 8192 characters, while the reads range from 64 to 4096 characters. The second evaluation, instead, consider a widely used dataset in the State of the Art, called *10s* [2].

4.2 Experimental Evaluation

Figure 1 depicts the throughput tendency for our analyzed datasets. In the first part of the plot, we observe that, employing our *dynamic memory swap*, we attain a performance improvement of approximately 13%, when aligning sequences of 64 and 128 characters in length. Then, with longer sequences, we observe a considerable increase in performance, reaching the peak of 75 GCUPS analyzing the 256-512 dataset. Tailoring the dynamically allocated shared memory to the sequences actually processed by the device allows optimizing the block management on the GPU, as more blocks can be executed in parallel on the same Streaming Multiprocessor (SM). We recreate the corner case for currently-available solutions with the dataset *320-640* (red vertical line in Figure 1). Indeed, State-of-the-Art GPU solutions can process alignments up to this size [5]. We overcome this limitation by still exploiting shared memory, improving the usability and performance of the accelerator. After this threshold, the throughput remains high as the efficient shared memory management achieved with the dynamic memory swap improves GPU utilization. The subsequent decrease in performance is caused by the increased amount of shared memory required by each alignment that reduces the number of blocks the GPU can spawn on a single SM. Finally, focusing on the last part of the plot, falling back to global memory, as in the 1024-2048 dataset case, allows us to support longer sequences whose memory requirements exceed the shared memory available for the single block. Providing support in such scenarios becomes particularly relevant considering the prominent use of third-generation, or long-read, sequencing technologies capable of producing sequences of thousands of characters. This evaluation provides an overview of the performance scaling of our accelerator, depending on sequence length. Nevertheless, real-life datasets contain different combinations of alignments, creating a hybrid scenario, where the computation of the PairHMM on short sequences overlaps with the one on longer ones allowing reaching high performance.

Table 1: Evaluation with *'10s'* dataset

Design	Technology Node [nm]	Avg. Execution Time [ms]	Speedup w.r.t Java baseline
Java baseline [2]	N/A	10800	1×
C++ version [2]	N/A	1267	9×
Intel Xeon AVX, 1 core [2]	N/A	138	78×
Intel Xeon 24 cores [2]	N/A	15	720×
NVIDIA K40 [2]	28	70	154×
NVIDIA Tesla V100 [2, 5]	12	27.1	399×
NVIDIA Tesla V100 [5]	12	13.8	783×
Intel Stratix V [7]	28	8.3	1301×
Intel Stratix V [4]	28	5.3	2038×
AMD Xilinx KU3 [8]	20	5	2160×
Intel Arria 10 [7]	20	2.8	3857×
Intel Arria 10 [4]	20	2.6	4154×
Intel Arria 10 (128 PE) [10]	20	2.2	4909×
NVIDIA Tesla V100	**12**	**1.3**	**8154×**

Table 1 shows the performance of multiple State-of-the-Art solutions when aligning sequences of the *10s dataset*. Results are reported over the Java baseline, reflecting the currently-used tool for genomics analyses. Our solution is able to achieve a 8154× over the baseline, outperforming all the hardware-accelerated solutions. More specifically, our apporach delivers more than 10× improvement on the best-performing GPU solution, and 1.69× speedup over the best FPGA design.

5 CONCLUSIONS

This work presented a high-performance GPU accelerator for the PairHMM algorithm featuring a novel methodology, the *dynamic memory swap*, to provide support for long sequences, overcoming the limitations of State-of-the-Art solutions. Experimental results prove the effectiveness of our approach, showing 8154× and 1.6× performance improvements against the software baseline and the fastest hardware-accelerated solution, respectively, when executing on an NVIDIA V100.

ACKNOWLEDGMENTS

This work has been partially supported by Compiler Frameworks and Hardware Generators for Innovative US Government Design. The Authors would like to thank Oracle Research Program for the Oracle Cloud Credits.

REFERENCES

[1] Subho S Banerjee, Mohamed El-Hadedy, Ching Y Tan, Zbigniew T Kalbarczyk, Steve Lumetta, and Ravishankar K Iyer. 2017. On accelerating pair-HMM computations in programmable hardware. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.

[2] Mauricio Carneiro. 2013. Accelerating variant calling. In *Broad Institute, Intel Genomic Sequencing Pipeline Workshop, Powerpoint Presentation, Mount Sinai*.

[3] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. 2010. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research* 38, 6 (2010), 1767–1771.

[4] Sitao Huang, Gowthami Jayashri Manikandan, Anand Ramachandran, Kyle Rupnow, Wen-mei W Hwu, and Deming Chen. 2017. Hardware acceleration of the pair-HMM algorithm for DNA variant calling. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 275–284.

[5] Enliang Li, Subho S Banerjee, Sitao Huang, Ravishankar K Iyer, and Deming Chen. 2021. Improved GPU Implementations of the Pair-HMM Forward Algorithm for DNA Sequence Alignment. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 299–306.

[6] Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* 48, 3 (1970), 443–453.

[7] Chris Rauer and N Finamore. 2016. Accelerating genomics research with opencl and fpgas. *Altera, Now Part of Intel, Tech. Rep* (2016).

[8] Davide Sampietro, Chiara Crippa, Lorenzo Di Tucci, Emanuele Del Sozzo, and Marco D Santambrogio. 2018. Fpga-based pairhmm forward algorithm for dna variant calling. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 1–8.

[9] Temple F Smith, Michael S Waterman, et al. 1981. Identification of common molecular subsequences. *Journal of molecular biology* 147, 1 (1981), 195–197.

[10] Pengfei Wang, Yuanwu Lei, and Yong Dou. 2019. Pair-HMM accelerator based on non-cooperative structure. *IEICE Electronics Express* (2019), 16–20190402.