SYCLOPS

Deliverable 5.4 – SYCL genomics acceleration library

GRANT AGREEMENT NUMBER: 101092877

Project acronym:	**SYCLOPS**
Project full title:	**Scaling extreme analYtics with Cross architecture acceLeration based on OPen Standards**
Call identifier:	**HORIZON-CL4-2022-DATA-01-05**
Type of action:	**RIA**
Start date:	**01/01/2023**
End date:	**31/12/2025**
Grant agreement no:	**101092877**

D5.4 - SYCL genomics acceleration library

Executive Summary: This deliverable details the work accomplished within WP5, “Task 5.4: SYCL-GAL Library”, focusing on the SYCL-GAL library developed to accelerate secondary genomic analysis. The library provides a SYCL-based implementation of key algorithms designed for performance-portable execution of genomic data analysis on heterogeneous hardware. The work targeted two primary computational bottlenecks identified in standard genomic pipelines, namely, post-alignment data preparation and the pair Hidden Markov Model (pairHMM) in variant calling. Within task 5.4, significant progress was achieved in both fronts.

WP:	5		
Author(s):	Nimisha Chaturvedi, Eugenio Marinelli, Aleksander Illic		
Editor:	Raja Appuswamy		
Leading Partner:	ACC		
Participating Partners:	EUR, INESC		
Version:	1.0	**Status:**	Draft
Deliverable Type:	R	**Dissemination Level:**	PU
Official Submission Date:	06-Oct-2025	**Actual Submission Date:**	30-Sep-2025

Disclaimer

The SYCLOPS consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	EURECOM	EUR	**FR**
2	INESC ID - INSTITUTO DE ENGENHARIADE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC	**PT**
3	RUPRECHT-KARLS-UNIVERSITAET HEIDELBERG	UHEI	**DE**
4	ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE	CERN	**CH**
5	HIRO MICRODATACENTERS B.V.	HIRO	NL
6	ACCELOM	ACC	FR
7	CODASIP S R O	CSIP	CZ
8	CODEPLAY SOFTWARE LIMITED	CPLAY	UK

Document Revision History

Version	Description	Contributions
0.1	Structure and outline	**EUR**
0.2	Background, use case and motivation	**ACC**
0.3	Post-alignment contribution	**EUR**
0.4	pairHMM contribution	**INESC**
1.0	Final draft	**EUR**

Authors

Author	Partner
Nimisha Chaturvedi	**ACC**
Eugenio Marinelli	**EUR**
Aleksander Ilic	**INESC**

Reviewers

Name	Organisation
Aleksandar Ilic	INESC
Vincent Heuveline	UHEI
Stefan Roiser	CERN
Nimisha Chaturvedi	ACC
Martin Bozek	CSIP

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of Contents

Executive Summary

This deliverable details the work accomplished within WP5, “Task 5.4: SYCL-GAL Library”, focusing on the SYCL-GAL library developed to accelerate secondary genomic analysis. The library provides a SYCL-based implementation of key algorithms designed for performance-portable execution of genomic data analysis on heterogeneous hardware. The work targeted two primary computational bottlenecks identified in standard genomic pipelines, namely, post-alignment data preparation and the pair Hidden Markov Model (pairHMM) in variant calling. Within task 5.4, significant progress was achieved in both fronts.

1. **Post-alignment preprocessing**: Since the standard GATK pipeline is CPU-based and inefficient due to disk I/O dependency and use of Java, SYCL-GAL developed a fully SYCL-based pipeline designed to run efficiently on GPUs. When benchmarking against the standard CPU-based GATK implementation, SYCL-GAL achieved an end-to-end speedup of 8× faster for the post-alignment stages. The performance breakdown showed superior acceleration in every stage, with sorting being 43× faster, mark duplicates being 27× faster, and BQSR being 11× faster. When compared against the proprietary, state-of-the-art NVIDIA Parabricks pipeline (CUDA implementation), Parabricks was only 1.7× faster than the SYCL-GAL implementation.
2. **Variant calling:** Variant calling is dominated by the Pair Hidden Markov Model (pairHMM) algorithm, which accounts for over 70% of the execution time in the GATK HaplotypeCaller. SYCL-GAL proposed a novel parallelization approach, referred to as "Endeavor," which mathematically redefines the processing steps of pairHMM to expose previously unexplored levels of parallelism without sacrificing solution accuracy. The average speedups achieved by SYCL-GAL across various GPUs and datasets were substantial. On NVIDIA H100 GPU, we achieved an average speedup of 306.3x. On Intel Max 1100 GPU, an average speedup of 204.9x was achieved. On H100 GPU, a maximum speedup observed was 470× was achieved. Compared to CUDA-based state-of-the-art gpuPairHMM, SYCL-GAL solution achieved an average speedup between 1.2× and 1.51× across tested NVIDIA GPUs

All developments are publicly available, with SYCL-GAL library have been made publicly available as open source software on the central SYCLOPS Github page[1].

[1] https://github.com/SYCLOPS-Project-EU/sycl-gal

1 Introduction

Figure 1 shows the SYCLOPS hardware-software stack consists of three layers: (i) infrastructure layer, (ii) platform layer, and (iii) application libraries and tools layer.

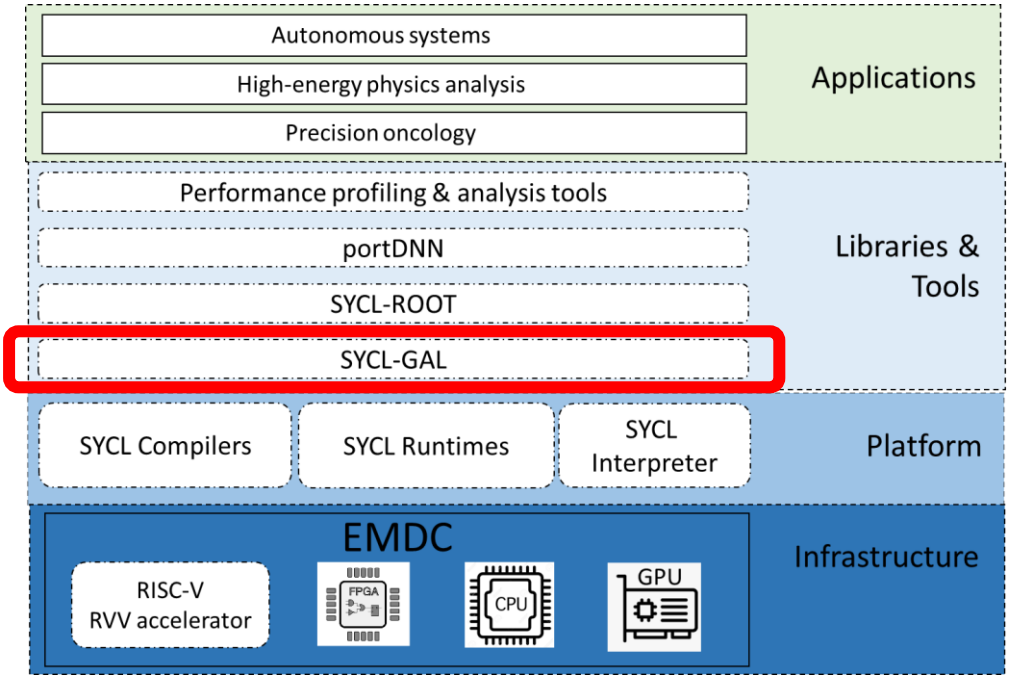

Figure 1. SYCLOPS architecture

Infrastructure layer: The SYCLOPS infrastructure layer is the bottom-most layer of the stack and provides heterogeneous hardware with a wide range of accelerators from several vendors.

Platform layer: The second layer from the bottom, the platform layer, provides the software required to compile, execute, and interpret SYCL applications over processors in the infrastructure layer. SYCLOPS will contain oneAPI DPC++ compiler from CPLAY, and AdaptiveCpp from UHEI. In terms of SYCL interpreters, SYCLOPS will contain Cling from CERN.

Application libraries and tools layer: While the platform layer described above enables direct programming in SYCL, the libraries layer enables API-based programming by providing pre-designed, tuned libraries for various deep learning methods for the PointNet autonomous systems use case (SYCL-DNN), mathematical operators for scalable HEP analysis (SYCL-ROOT), and data parallel algorithms for scalable genomic analysis (SYCL-GAL)

This deliverable covers the **SYCL-GAL** part of the stack as highlighted in Figure 1. Several developments have been made in the context of “WP5: Task 5.4 SYCL-GAL Library”. This deliverable is a summary of this work done.

This deliverable is structured as follows. Section 1 of this deliverable provides a high-level overview of the overall SYCLOPS architecture and positions this deliverable with respect to both components in the SYCLOPS stack and WP/tasks in the work plan. Section 2 gives the background on the genomic data analysis use case of SYCLOPS and motivates the need for SYCL-GAL. Section 3 and 4 describes the core work done in the library.

2 Background: Use Case & Context

Use Case. Due to rapid advances in genomic sequencing technology, it is possible today to sequence an entire human genome for less than $1000 and quickly obtain genetic data on different molecular levels of the cell. However, in order to transform this genomic Big Data into actionable insights for precision medicine, one needs to perform a systematic integrated analysis of a variety of biological datasets. ACCELOM has developed reproducible multi-omics software pipelines that use statistical machine learning techniques to integrate multiple molecular datasets.

The key challenge faced by ACCELOM is that its pipelines depend are designed for a CPU-based deployment, as they depend on the GATK pipeline developed by Broad Institute that has become the gold standard for genomic data analysis. Figure 2 below shows the GATK pipeline developed by Broad Institute that has become the gold standard for genomic data analysis, and hence, forms the core part of secondary data analysis performed by ACCELOM. As can be seen, samples containing patient's DNA is sequenced using Illumina sequences to get *Reads.* These reads are then subjected to a series of preprocessing steps in order to get them ready for variant analysis. These stages involve (i) aligning the reads to a reference genome, (ii) eliminating duplicates in reads, and (iii) readjusting base quality scores. The last two stages (duplicate elimination and base quality adjustment) are often referred to as "post-alignment preprocessing". Following preprocessing, we move into variant discovery, where various tools, like the HaplotypeCaller, are used to identify variants. Finally, this is followed by variant filtration, annotation, and further downstream analysis by ACCELOM's machine learning models.

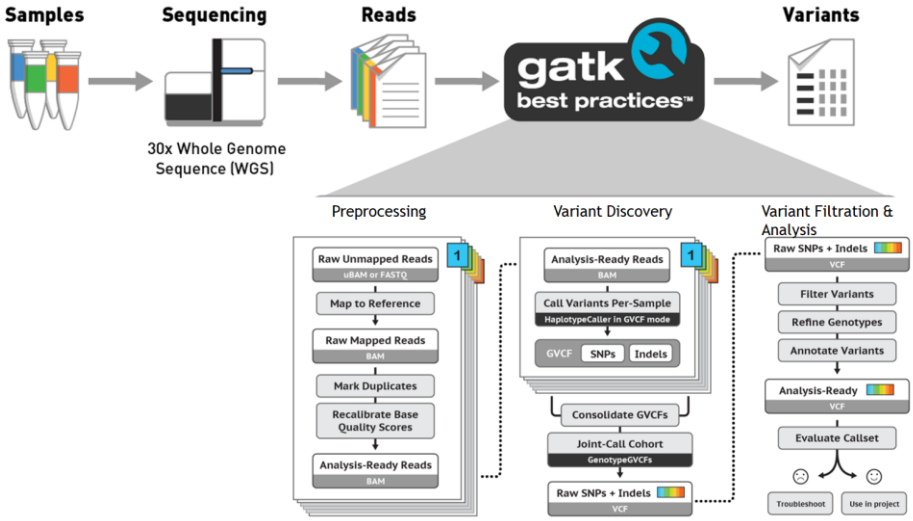

Figure 2. GATK Pipeline

Performing integrated analysis is a very computationally-intensive process that involves running the GATK pipeline for each data set, followed by the execution of machine learning models in a tertiary stage. As the stock GATK pipeline is CPU based, ACCELOM is bottlenecked on this CPU-based secondary analysis of genomic data. Hence, as the amount of genomic data grows exponentially, it becomes increasingly difficult for ACCELOM's CPU-based pipelines to perform such an analysis in a timely and cost-efficient manner.

The goal of "Task 5.4: SYCL-Genomics Acceleration Library" in WP5 is to develop a SYCL-based, open-source Genomics Acceleration Library that contains the algorithms required for developing an SYCL-based, accelerated genomics pipeline. During the first phase of the project, we had analysed ACCELOM's pipelines and identified two important stages of genomic analysis that are key bottlenecks. The first is post-alignment preprocessing. In this task, the aligned sequencing reads are subjected to a series of sorting, filtration, and quality estimation steps as mentioned earlier. The second bottleneck is a pair Hidden Markov Model (pairHMM) algorithm in the variant analysis stage that makes variant calling computationally very intensive. Thus, we focused on developing key algorithms in SYCL-GAL for accelerating these two stages in SYCLOPS. The rest of this document first describes our work on accelerating post-alignment preprocessing (Section 3) and variant calling (Section 4).

3 Acceleration of Post-Alignment Preprocessing

After aligning the raw DNA sequences against a reference genome, the resulting alignment file needs to undergo additional data preparation stages. In fact, in order to perform any insightful analysis (variant discovery, etc.) it is necessary to remove any source of bias. The three operations to be performed are Sorting, Marking duplicates and BQSR. Sorting is mainly necessary as a requirement of the marking duplicates stage, as it requires that reads are sorted by alignment coordinates with respect to the reference genome. Once the reads are sorted, mark duplicates aims to mark duplicate reads, i.e. multiple sequencing reads originating from the same DNA fragment, introduced during sequencing; BQSR has the objective to correct the systematic bias due to the instrument used to read the DNA strands during sequencing.

The standard framework used worldwide to perform post-alignment data preparation is GATK, developed by the Broad Institute. It is a suite of tools, mainly Java programs, each of which performs one different stage of the pipeline. Despite being widely adopted and representing the gold standard for genomics analysis, it comes with several limitations from the computational point of view, which make the analysis exceptionally time consuming, as we report in the results below. The main reason is that between each stage, GATK tools heavily rely on disk to store and read the results. Another limiting factor is the programming language, Java, which can underperform with respect to high performance languages such as C++ for compute-intensive tasks.

In order to overcome the low performance of these tools, we developed a SYCL-based Post-Alignment Data Preparation pipeline in SYCL-GAL. The main advantages introduced by SYCL-GAL are the fact that it is entirely written in SYCL, and therefore it can run on any hardware architecture (CPU, GPU). The design of the different stages is thought to run on GPU, allowing us to speed up execution using GPU and CPU synergistically.

3.1 Data Structures

Rewriting the stages originally designed for CPU in SYCL is not enough to obtain good performance on GPU. Therefore, a redesign of the algorithms is necessary to fully exploit the device. Normally, reads post alignment are stored in a file format called SAM or its binary representation BAM. The SAM/BAM file stores data as an Array of Structures (AoS), which means that every read and its attributes are stored together (i.e. in a line), and multiple lines make an array. This data layout is not the most efficient for GPU, as it leads to low cache utilization: fields across different reads are not guaranteed to be adjacent in memory, and this makes GPU execution particularly inefficient.

Therefore, the first change we introduce with respect to. GATK tools is the so-called Structure of Arrays (SoA) layout. This layout consists in storing the same fields for all the reads in flat vectors, and keeping one vector per field. In addition to better cache utilization, an advantage of this approach is the fact that we can selectively pick only the fields needed for a given stage. In fact, not every field of the read is needed in every stage. This makes it easy to copy only those fields to device memory, as we just need to transfer the relevant field vectors.

3.2 Sorting

Sorting is the first post-alignment stage we redesigned to suit the hardware accelerated SYCL-GAL. As we mentioned above, sorting is used mainly as a prerequisite for the later mark duplicates stage. We kept the stage on CPU, using multithreading. Due to the SoA layout we structured the algorithm in two phases. First, we create a sorted index based on the values of our SoA. This index represents the relative order of each read and therefore its final position in the vector. In the second phase, we scatter the SoA elements based on the computed index.

3.3 Mark Duplicates

Mark duplicate stage is the second stage we optimized. The idea is that if two reads (or read pairs) are mapped to the same positions in the reference genome with the same orientation, they likely originate from the same DNA fragment and are therefore considered duplicates. For paired-end reads, both mates' alignment positions and orientation are taken into account when grouping duplicates. The original GATK algorithm performs duplicate marking by storing a global hash map. The hash map keeps track of the read whose quality score (or summed base qualities) is the highest among all duplicates. The algorithm scans sequentially the input reads and updates the hash map once a better representative is found.

To make the algorithm suitable for GPU, we removed the hash map, as it can lead to low performance when the collision rate is too high (i.e., too many duplicates due to high sequencing coverage). The alternative we propose consists in scanning the input reads and emitting into a flat vector only the fields needed to identify duplicate sets together with the score of each read. Then we sort this vector by key (where the key is made of the mapping coordinates, the reference chromosome, and mate/orientation fields needed to define duplicate sets): all reads of the same duplicate set will then be in adjacent positions. Finally, we go over the vector and mark for each set the read with the highest score as "origin" and the reads with lower score as duplicates. Read mapping, sorting and marking of duplicates are expressed as SYCL kernels.

3.4 Base Quality Score Recalibration (BQSR)

Sequencing machines in addition to call the different nucleotides bases in a fragments, they also provides an estimates of the accuracy for each base. This is done by assigning a quality score (Phred) in logarithmic scale, indicating the probability that the base is wrong. For example, a quality score of Q30 for a base corresponds to an error probability of 0.001, which means that the sequencer is 99.9% sure that the base called is correct. Despite the probability of being wrong seems to be very low, the number is actually not negligible if we consider that we are dealing with billion of bases in a 30x genomes. Base quality scores are central to evaluating the strength of evidence for or against candidate variant alleles in short-variant discovery. If these scores are systematically biased, variant calling accuracy suffers. Such systematic errors can arise from multiple sources, including the biochemistry of library preparation and sequencing, flaws in the flow cell manufacturing process, or imperfections in the sequencing instrument itself. Fortunately, those are not random errors and therefore by gathering statistics across bases in the dataset with respect to various covariates (such as reported quality, sequencing cycle, and local base context), we can (1) construct an error model and (2) adjusts the reported quality scores of each base according to that model. This process is called **Base Quality Score Recalibration (BQSR).**

BQSR is a two-stage process. The gold standard tools used to perform this operation are provided by the GATK Secondary Analysis toolkit, as two Java programs: BaseRecabrator builds the error models and saves them in the form of tables; ApplyBQSR computes, for each base in every read, a new quality score derived from the tables built previously.

While GATK tools are widely used and accepted as gold standard, the fact that they can run on CPU only and that are written in Java makes the tools completely inefficient and time consuming. To overcome these bottlenecks, several acceleration efforts have been explored. FPGA-based implementations (e.g., [Lo et al., FCCM 2020]) demonstrate energy-efficient designs, while NVIDIA's Parabricks reimplements BQSR in CUDA as part of a fully GPU-accelerated secondary analysis pipeline. However, these solutions are tied to vendor-specific hardware stacks and lack portability, and to the best of our knowledge there is no prior work proposing a detailed design of these stages on GPU or open source GPU-accelerated solutions.

Therefore in this work, we developed a SYCL-based GPU implementation of BQSR that combines the performance advantages of GPU acceleration with the portability of a single-source C++ model. Our approach rethinks the original CPU algorithm using GPU-friendly map–reduce primitives (sorting and segmented reduction) to replace hash map updates, and leverages a columnar data layout to maximize memory throughput. This design allows us to exploit heterogeneous hardware (CPU + GPU) while maintaining broad portability across NVIDIA, Intel, and AMD platforms.

3.5 Evaluation

To validate our pipeline, we compared it against two frameworks currently available to perform the GATK secondary analysis stages (Sorting, Mark Duplicates and BQSR): **GATK** and **NVIDIA Parabricks**. NVIDIA Parabricks reimplements the GATK genomics pipeline end to end, from sequence alignment to BQSR (i.e., Aligner, Sorting, Mark Duplicates, and BaseRecalibrator), in CUDA, targeting NVIDIA GPUs. GATK, on the other hand, relies on the BWA-MEM algorithm to perform sequence alignment, and on its own suite of tools written in Java for the post-alignment stages. Each stage is a different standalone Java program.

We ran our pipeline on a 7 GB raw reads dataset provided by Parabricks, which after alignment generates a 20 GB SAM file. Benchmarks were performed on a server equipped with 256 GB RAM, a 64-core Intel(R) Xeon(R) Platinum 8454H CPU, and an NVIDIA L40S GPU available in our SYCLOPS EMDC.

By comparing the end-to-end execution time for SYCL-GAL, Parabricks, and GATK in the post-alignment stages, we observe that SYCL-GAL is 8× faster than the GATK implementation and 2.5× slower than the proprietary NVIDIA Parabricks. The main reason is that Parabricks, having integrated the aligner into the pipeline, takes as input a FASTQ file (raw sequenced reads), which is ~5× smaller than the alignment file (SAM/BAM). Our pipeline instead must read and parse a much larger SAM file, which negatively affects the end-to-end runtime. If we look at the breakdown of the computational stages only (Mark Duplicates and BQSR), comparing SYCL-GAL with Parabricks, we find that Parabricks is only 1.7× faster than our SYCL implementation. Furthermore, when comparing the breakdown of SYCL-GAL with GATK, we see that we outperform GATK in Sorting, Mark Duplicates, and BQSR by factors of 43×, 27×, and 11×, respectively.

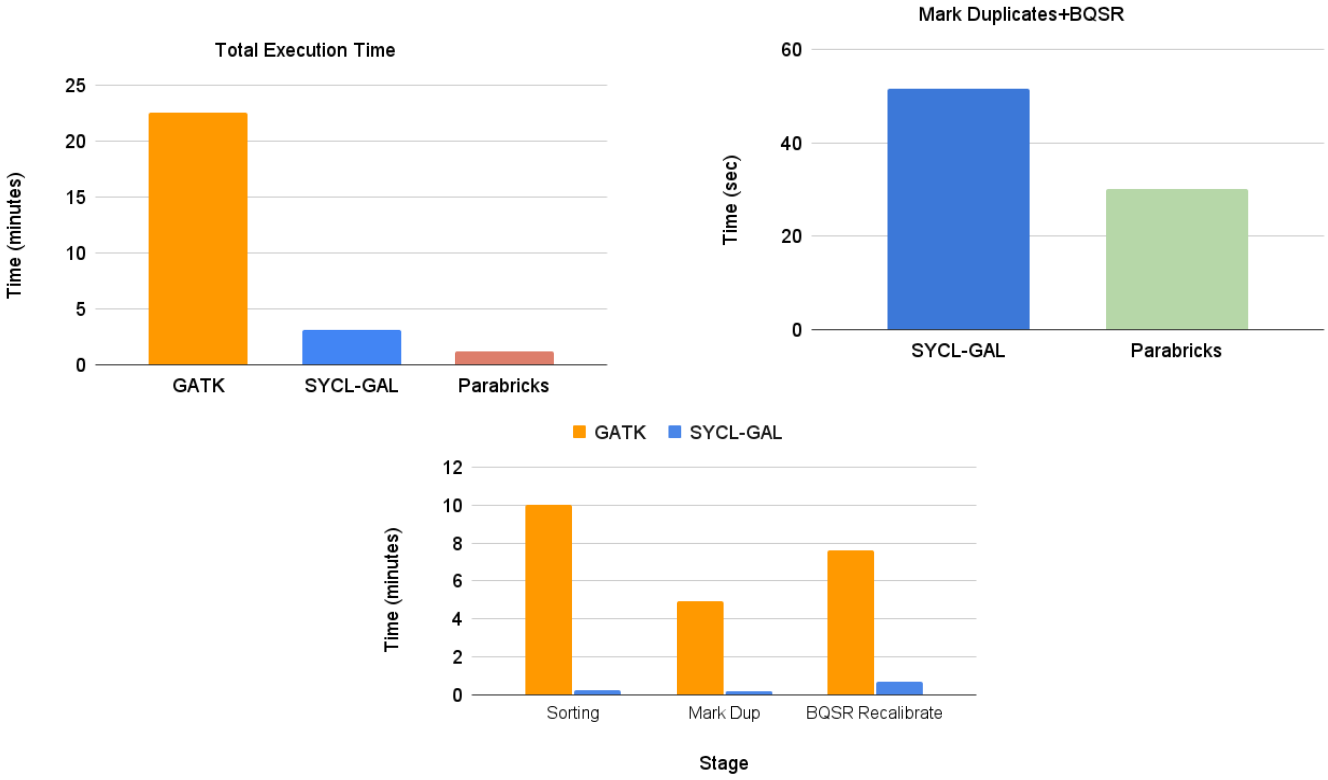

Figure 3. SYCL-GAL versus NVIDIA Parabricks

The target KPI for SYCLOPS is to achieve a 2x improvement in execution time for our use cases. Our results clearly show that SYCL-GAL goes well beyond this KPI. Our results also show that a SYCL-based implementation can provide performance competitive with CUDA-based counterparts. We believe these results are very promising as they suggest that open, vendor-neutral, cross-architecture acceleration is possible for data-intensive domains like genomics.

4 Acceleration of pairHMM in Variant Calling

Variant calling analyses the existence of genetic dissimilarities between a patient's genome and a reference genome. This bioinformatics pipeline is of the utmost importance to identify genetic differences and understand their correlation to complex diseases, which has a significant impact in precision medicine, pharmacogenomics, and evolutionary biology. As a consequence, specific toolkits have been developed to process genetic data for variant calling, with the Genome Analysis Toolkit (GATK) HaplotypeCaller being one of the most widely used in genomic analysis.

The typical workflow of variant calling in GATK HaplotypeCaller (one of the most widely used tools in genomic analysis) follows three steps. Given a dataset with read sequences from a patient, the first step is to generate a set of candidate haplotypes by assembling the reads. The second step is to calculate the probability that a read was generated by a given haplotype. To this end, the Pair Hidden Markov Model (PairHMM) algorithm aligns the read sequence data and the candidate haplotypes, calculating alignment likelihoods for each candidate and selecting the haplotypes with the highest likelihood. Finally, a Bayesian model is applied to the computed likelihoods to determine the most probable genotype at a given read (if it is equal to a reference genome or a variant).

Of these three steps, the PairHMM algorithm is the main bottleneck, responsible for over 70% of the execution time in genome-scale datasets, and is therefore the focus for hardware acceleration efforts. However, the PairHMM algorithm is not trivial to accelerate and the main reasons for this are two-fold. First, PairHMM scales quadratically in time and memory with the sequence size (as it is based on computing matrix cells), which makes it prohibitive for sequences generated with long-read sequencing technologies. While the memory requirements can be reduced to scale linearly with the sequences, the same cannot be done for time requirements.

Second, the algorithm uses dynamic programming, which hinders the opportunities to parallelize the algorithm due to dependencies in the data flow. The PairHMM algorithm is based on processing three matrices (M, I, and D), where the anti-diagonals are independent, which has been explored for parallelism in GPUs. However, the anti-diagonals are of variable length (i.e., the workload is irregular), which hinders performance.

This is a particular gap that this work intends to close by proposing a novel parallelization approach for the PairHMM algorithm on GPUs that fully leverages the hardware resources and scales to long DNA sequences. The main contributions of this work are the redefinition of the processing steps of the PairHMM algorithm to expose previously unexplored levels of parallelism with no loss in solution accuracy and scalability to diverse genomic datasets with short- and long-read sequences.

Figure 4 provides a high-level description of the proposed approach. To improve performance, the core operations of the PairHMM algorithm are redefined through mathematical manipulation to expose row-level parallelism of the matrices, which better fits the GPU hardware. Read-haplotype pairs are processed at the warp-level, with each thread calculating several elements of a row in parallel. After the last row is calculated, the output likelihood is calculated by reduction. Since the implementation processes a read-haplotype pair at the warp-level, many sequence pairs can be calculated concurrently, as a kernel launch can be done with multiple thread blocks, each with multiple warps. The number of sequence pairs to process can be adjusted to achieve maximum throughput of the algorithm.

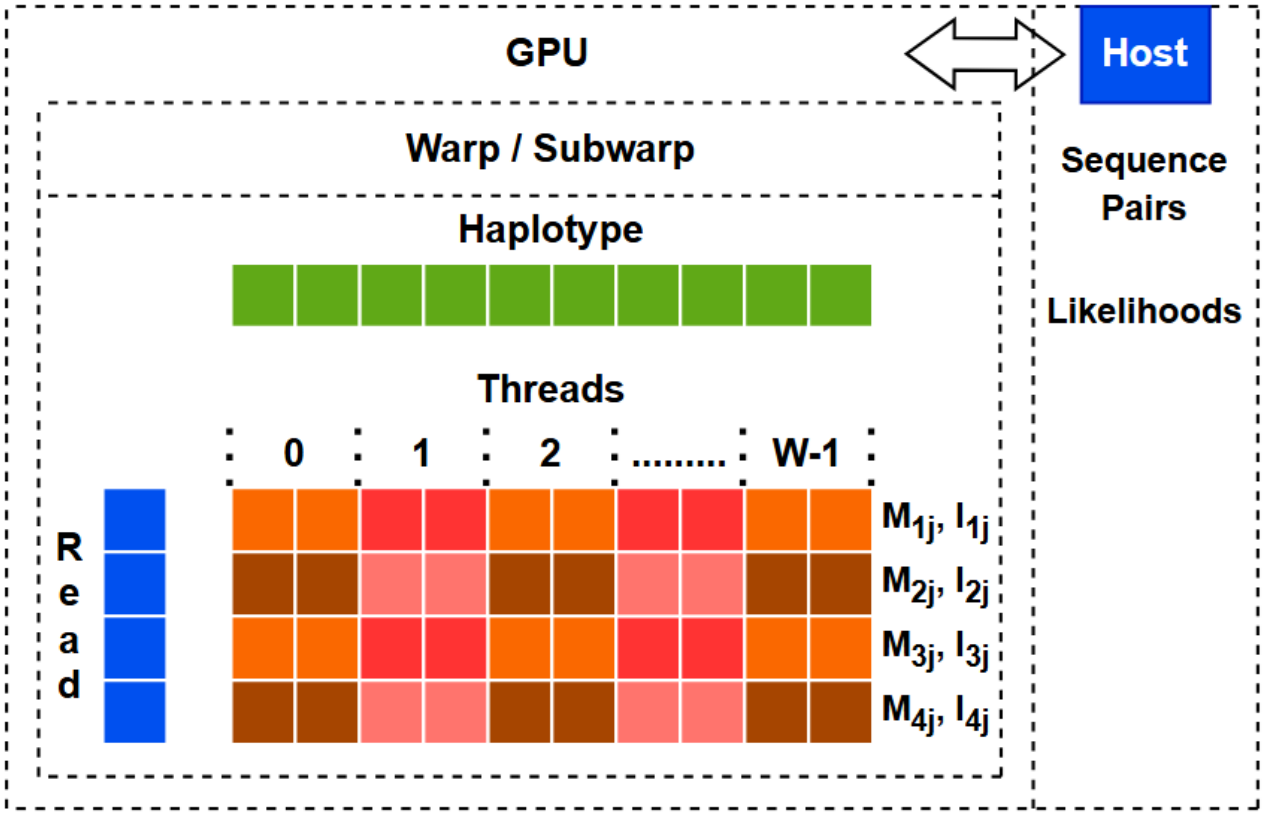

Figure 4: Warp-Based Implementation of PairHMM.

4.1 Experiments

The reported experiments have been performed on 5 GPUs from all major vendors, i.e., NVIDIA V100, A100, and H100 GPUs, using the CUDA 12.0 toolkit, AMD MI250X, using ROCM 6.2.2, and Intel Max 1100 GPU, using SYCL from Intel oneAPI 2024.2.1. To fully examine the capabilities of the proposed approach in datasets generated with short-read and long-read sequencing technologies, datasets from the GIAB consortium are used, namely the NA12878}, NA24695, NA24631, and NA24149 datasets. For the GIAB datasets, the proposed approach is tested on all GPUs and compared with gpuPairHMM (as it is the current best PairHMM implementation for GPU) and GATK HaplotypeCaller, which is capable of processing long-read datasets, on an Intel Sapphire Rapids CPU (with AVX-512 instructions).

4.1.1 Scalability

Figure 5 displays the TCUPS evolution in the H100 GPU as the input read-haplotype length increases. Three curves are displayed, two for Endeavor in FP32 and FP64, and one for the state-of-the-art approach, gpuPairHMM, in FP32 (gpuPairHMM does not have a FP64 implementation, and therefore cannot be compared to Endeavor for this precision). Given the warp size of 32 for NVIDIA GPUs, a single warp can process read-haplotype pairs of length up to 1024 with FP64 and up to 2048 with FP32 (blue area). For long read-haplotype pairs, processing is done at the level of multiple warps, leveraging shared memory to store necessary values across warps (orange area) and at the level of multiple thread blocks for very long reads (green area). As gpuPairHMM only runs for sequence lengths up to 1024, its curve is not represented for longer read-haplotype pairs.

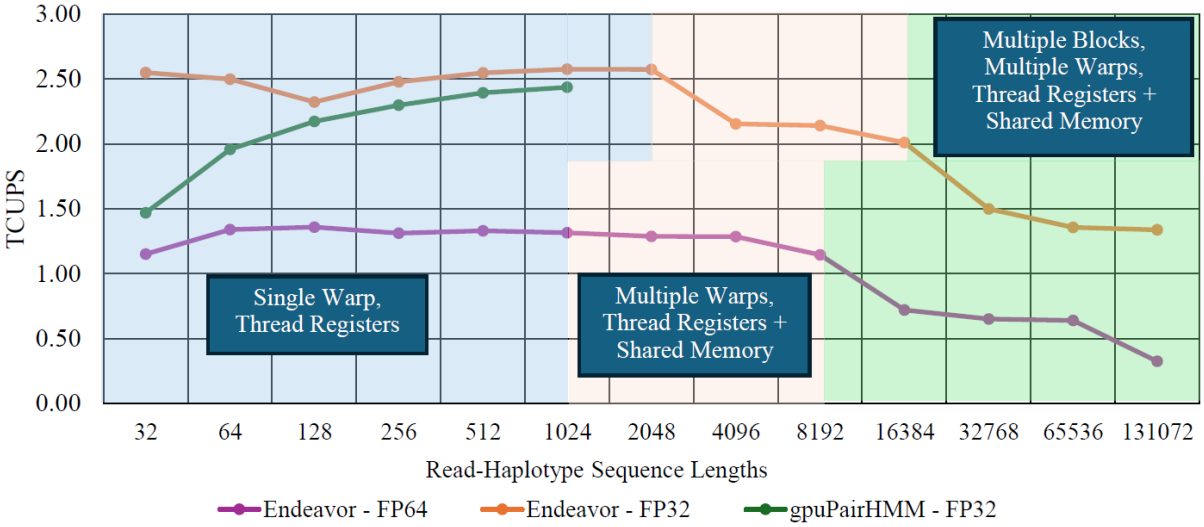

Figure 5: TCUPS evolution in H100 for Endeavor (FP32, FP64) and gpuPairHMM (FP32).

In FP32, gpuPairHMM's peak throughput is 2.43 TCUPS, while our approach achieves 2.57 TCUPS, which represents a ~6% increase for a sequence length of 1024. As the sequence length decreases, we achieve a larger performance gain over gpuPairHMM. For a sequence length of 32, gpuPairHMM achieves 1.47 TCUPS, while our solution achieves 2.02 TCUPS, a 37% increase in throughput. For read-haplotype lengths above 1024, the peak throughput decreases to 2.15 TCUPS (orange area) and 1.5 TCUPS (green area), Nevertheless, the achieved throughput is above 1.3 TCUPS, even for sequence lengths up to 131072, which is 2 orders of magnitude above what gpuPairHMM is able to process. For FP64, the proposed framework achieves a peak throughput of 1.36 TCUPS, decreasing to 1.29 TCUPS when using multiple warps and to 0.72 TCUPS when using multiple thread blocks.

To further demonstrate the efficiency of our solution in using the GPU's resources, Figure 6 displays H100's roofline, generated with CARMTool, to evaluate our approach (for sequence lengths of 32, 2048, 16384, and 131072) and gpuPairHMM (for sequence lengths of 32 and 1024). The roofline shows that gpuPairHMM is closer to the memory-bound region (i.e., the main limitations come from the memory subsystems, which hinders possible performance improvements), while the proposed approach, as a redefined PairHMM approach, is compute-bound (i.e., the main bottleneck is the amount of compute resources in the hardware).

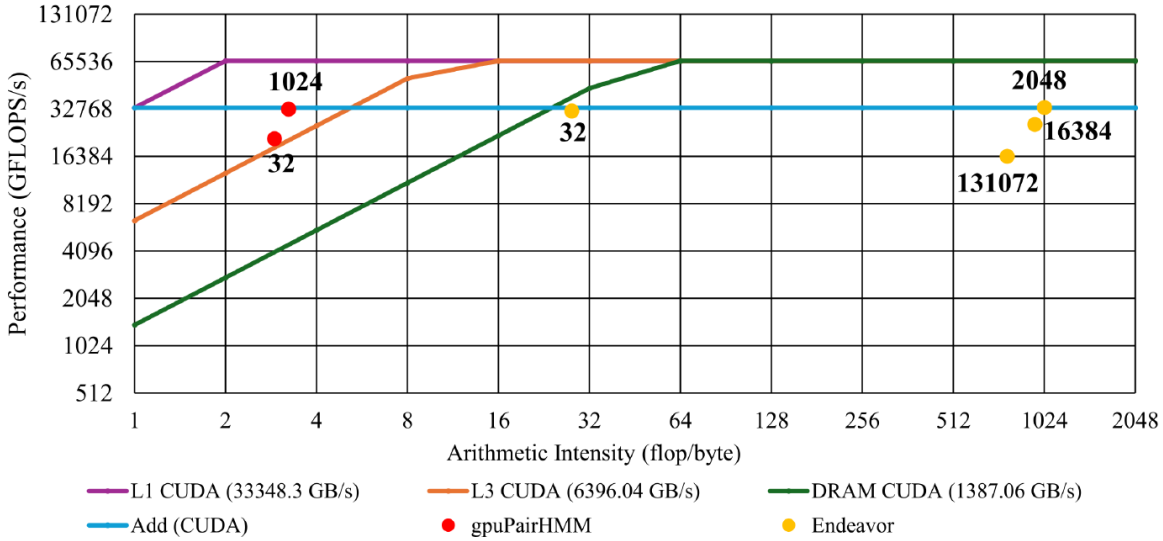

Figure 6: CARM Roofline for Endeavor and gpuPairHMM (FP32)

4.1.2 State-of-the-art Comparison

Experiments on GIAB datasets from seven different sequencing technologies are performed to evaluate the proposed solution's flexibility and performance on short-read datasets (sequence lengths in the range of 250-800) and long-read datasets (sequence lengths > 10000). For short-reads, four different datasets are considered, generated by Illumina, SoLiD, BGI-SEQ500, and Ion Torrent, with varying number of sequence pairs to process. For long-reads, three different datasets are evaluated, generated by Chromium Long Ranger, PacBio, and ONT sequencing technologies, with the last one exhibiting the sequences with longest length (12121 basepairs). The BGI-SEQ500, ONT and Chromium datasets refer only to Chromosome 22, while the remaining datasets refer to the complete human genome.

Figure 7 provides the speedups achieved over GATK HaplotypeCaller's AVX-512 implementations on all datasets across all GPUs. When compared to the AVX-512, the best PairHMM speedups occur for the ONT dataset on all GPUs (167.3x (V100), 215x (A100), 470x (H100), 195.8x (MI250x), and 275.8x (Max 1100)). The average speedups are 115.6x (V100), 148.4x (A100), 306.3x (H100), 180.6x (MI250X) and 204.9x (Max 1100) when compared to AVX-512. In addition to the comparison with the GATK HaplotypeCaller's PairHMM implementation, the impact of our approach's integration in the GATK pipeline is also evaluated in Figure 8. Analyzing the results on GPUs (orange bars), our solution achieves the best GATK pipeline speedups for the Ion Torrent dataset (2.31x). This is to be expected given the percentage that the PairHMM algorithm occupies in the total computation time (blue line) for this particular dataset (57.28%), where GPU-based approaches have a more noticeable impact. For datasets where the PairHMM percentage is much smaller, the pipeline speedup is not as prominent. As an example, in the SoLiD dataset (6.32%), the speedup is 1.07x (when compared to AVX-512).

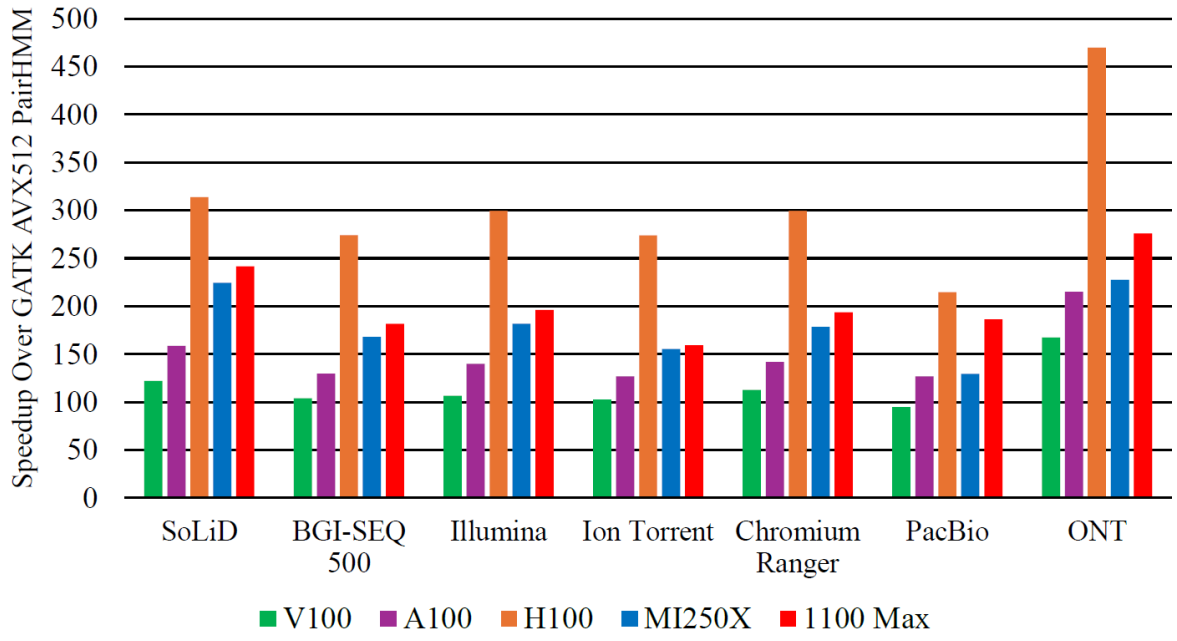

Figure 7: GATK AVX-512 Comparison

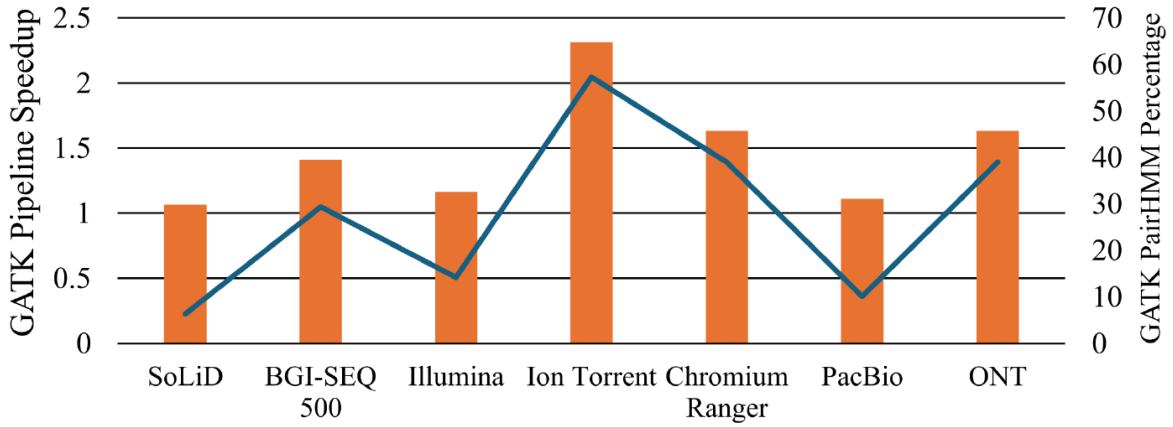

Figure 8: GATK pipeline speedup using Endeavor (orange bars) and GATK PairHMM percentage (blue line)

Figure 9 provides the speedups achieved over gpuPairHMM with the proposed framework on the SoLiD, BGI-SEQ500, and Chromium Ranger datasets, in addition to two datasets provided in the gpuPairHMM paper: a subset of the NA12878 dataset from the 1000 Genomes project and the 10s dataset replicated 4096x. While the maximum sequence length for the Ion Torrent and Illumina datasets is within range to use gpuPairHMM, this framework throws an out-of-memory (OOM) error due to the size of these datasets (which have over 10x the number of read-haplotype pairs to process when compared to SoLiD, BGI-SEQ500, and Chromium Ranger). The results show that the proposed approach can outperform gpuPairHMM on all datasets on all tested NVIDIA GPUs, with the smallest speedups achieved for the Chromium Ranger dataset (1.3x (V100), 1.06x (A100), and 1.11x (H100)) and the best performance on the SoLiD dataset (1.95x (V100), 1.37x (A100), and 1.5x (H100)). The proposed solution achieves an average speedup of 1.51x (V100), 1.2x (A100), and 1.28x (H100), while also running datasets that gpuPairHMM cannot process.

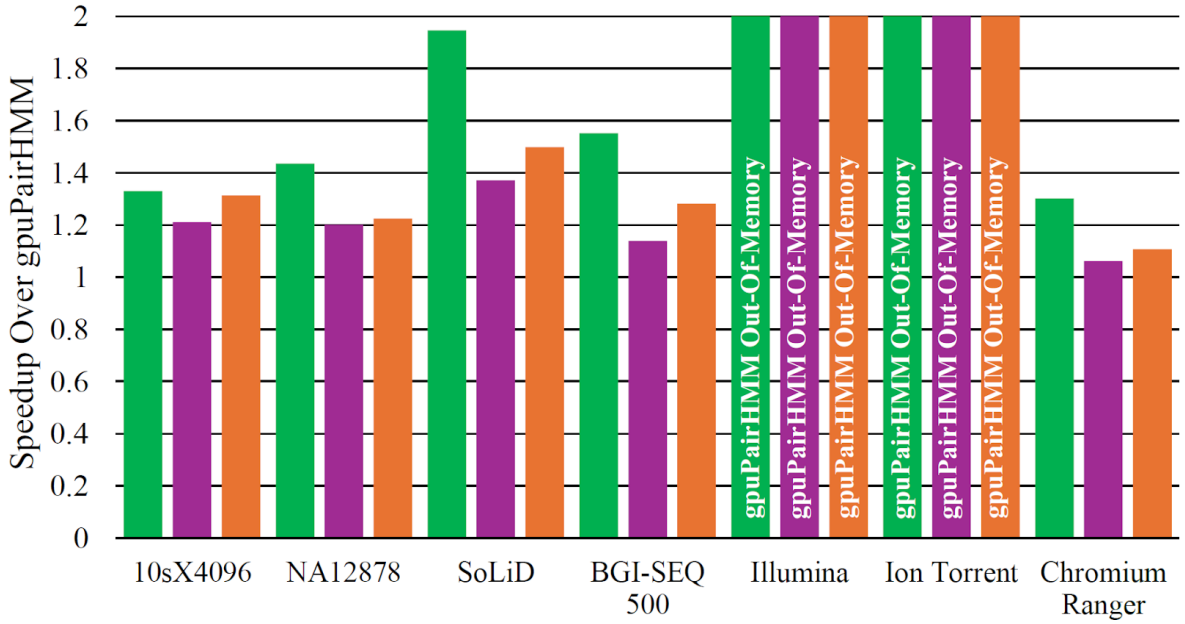

Figure 9: Speedup of Endeavor over gpuPairHMM (higher is better)

5 Conclusion

This deliverable concludes the work performed in “Task 5.4: SYCL Genomics Acceleration Library”. The main achievement is the development of the SYCL-GAL library , which provides a SYCL-based implementation of key algorithms designed to enable performance-portable acceleration of secondary genomic analysis on heterogenous hardware. The work successfully addressed the two primary computational bottlenecks identified in genomic analysis pipelines

1. Post-alignment data preparation: Accelerated stages including Sorting, Mark Duplicates, and Base Quality Score Recalibration (BQSR).
2. Variant calling: Developed a novel parallelization approach for the Pair Hidden Markov Model (pairHMM) algorithm

All developments related to the SYCL-GAL library are publicly available as open source software on the central SYCLOPS Github page[2].

In the final phase of the project, we plan to publish our work in peer-reviewed venues. We will also be integrating SYCL-GAL library into the pipeline of our use case partner ACCELOM, deploy the updated pipeline on EMDC v2.0 described in D3.2, and perform end-to-end performance testing using publicly available genomic data to demonstrate the benefit that SYCLOPS brings about to ACCELOM.

[2] https://github.com/SYCLOPS-Project-EU/sycl-gal