GPU accelerated partial order multiple sequence alignment for long reads self-correction

19th IEEE International Workshop on High Performance Computational Biology,
May 18, 2020,
New Orleans, Louisiana USA

Francesco Peverelli:	**francesco1.peverelli@mail.polimi.it**	Steven Hofmeyr:	shofmeyr@lbl.gov
Lorenzo Di Tucci:	lorenzo.ditucci@polimi.it	Aydın Buluç:	abuluc@lbl.gov
Marco Domenico Santambrogio:	marco.santambrogio@polimi.it	Leonid Oliker:	loliker@lbl.gov
Nan Ding:	nanding@lbl.gov	Katherine Yelick:	kayelick@lbl.gov

IPDPS 2020 New Orleans

BERKELEY LAB
Lawrence Berkeley National Laboratory

POLITECNICO MILANO 1863

POLITECNICO MILANO 1863
NECST laboratory

Third generation sequencing

- provides much **longer reads** allowing more precise **contig and haplotype assembly** and **structural variant calling**

- the **error rate** of these sequences is **significantly higher (10-20%)** compared to their second generation counterparts (0.2%)

- therefore, **error correction** is included as a preliminary step in genome analysis

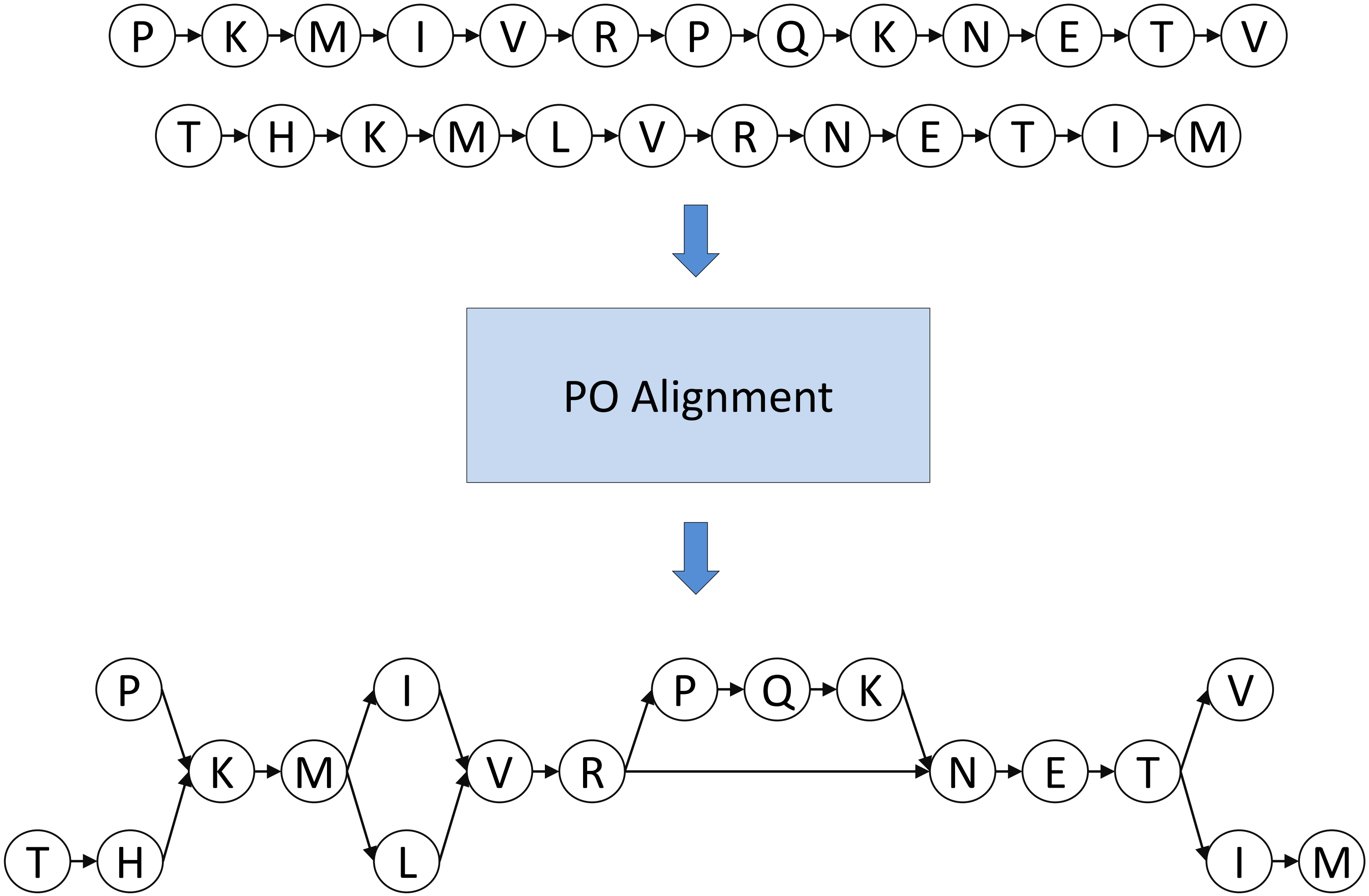
- many self-correction tools (e.g. RACON, CONSENT) rely on **Partial Order (PO) Multiple Sequence Alignment (MSA)** to identify the consensus sequences

Contributions

- A **GPU implementation** of the PO alignment algorithm that achieves up to **6.5x speedup** compared to the software version run on two 2.3 GHz **16-core Intel Xeon Processors E5-2698 v3** with 64 CPU threads
- An extension of the **Roofline model analysis** for **GPUs** presented in [1], to evaluate the performance of our implementation on the NVIDIA **Tesla V100**
- The **integration** of our kernel with **CONSENT**, a state of the art long read self-correction tool obtaining up to **8.5x speedup** of the error correction module

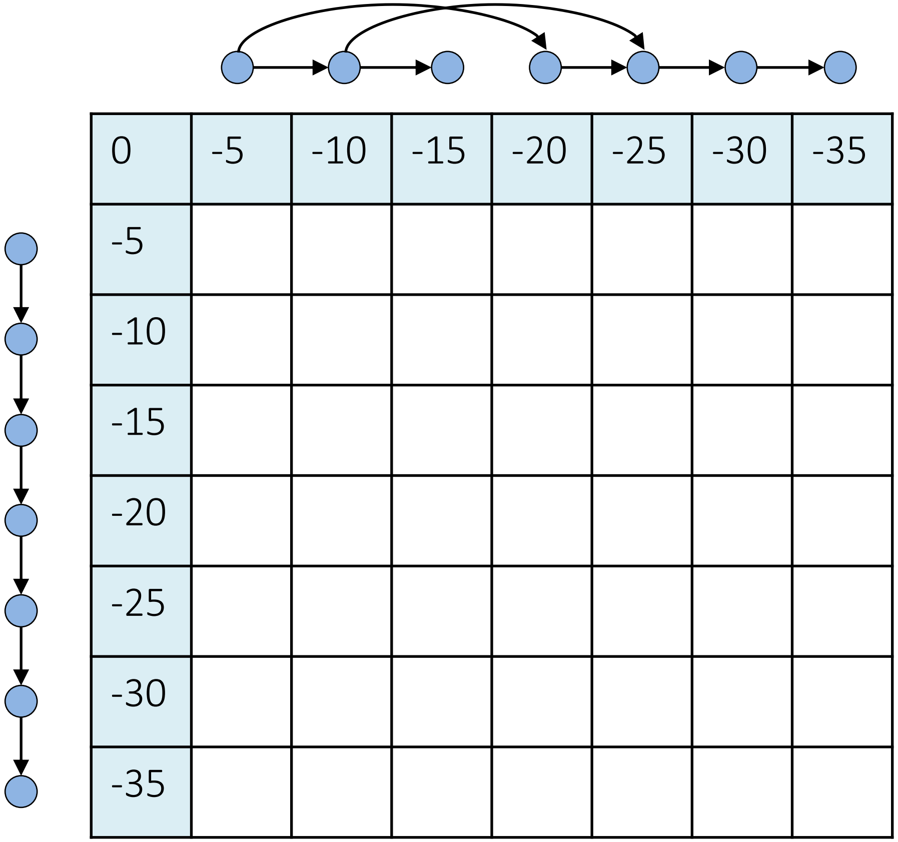
[1] N. Ding and S. Williams, "An instruction roofline model for gpus," 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2019.

Partial order graph alignment



PO Alignment

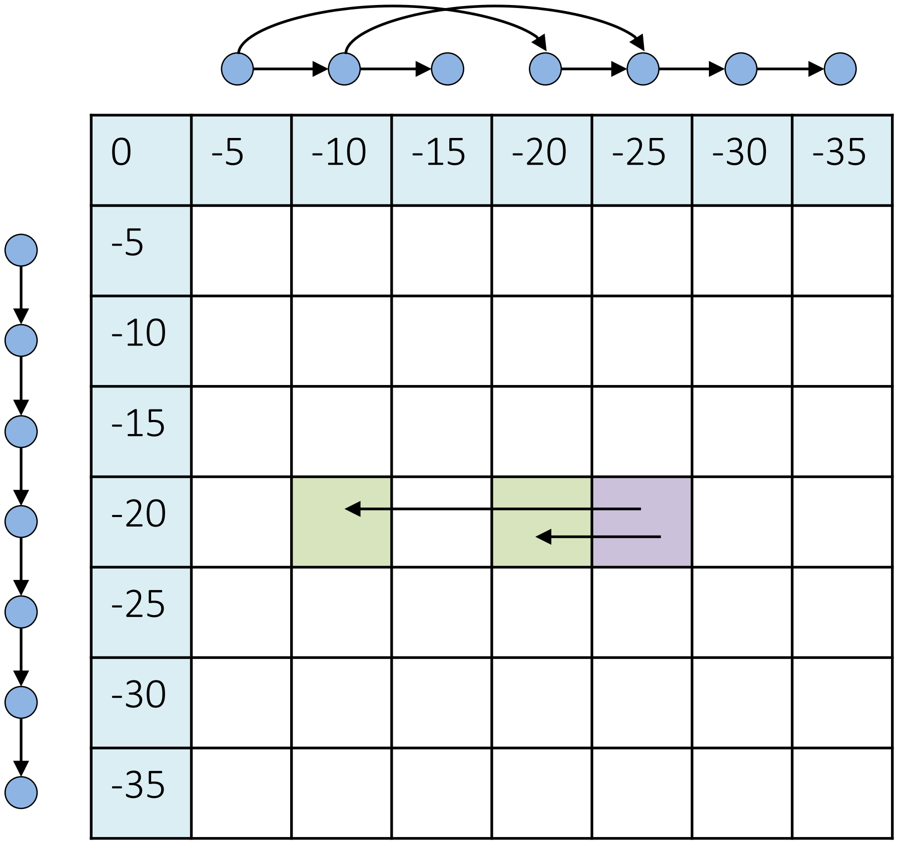
Partial Order Alignment



0	-5	-10	-15	-20	-25	-30	-35
-5							
-10							
-15							
-20							
-25							
-30							
-35							

Similarly to sequence alignment, a scoring matrix is used to indentify the optimal alignment
Between the PO graphs

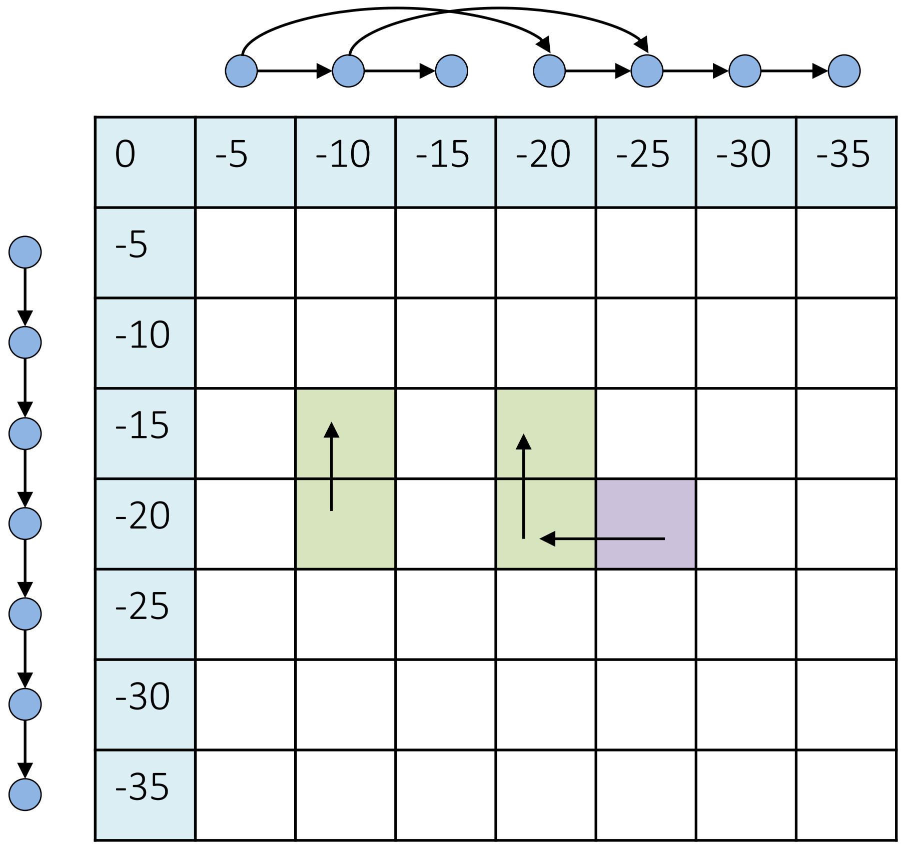
Partial Order Alignment



0	-5	-10	-15	-20	-25	-30	-35
-5							
-10							
-15							
-20							
-25							
-30							
-35							

🟪 Cell to score at current iteration
🟩 Scoring dependencies
← Dependency arc

Partial Order Alignment



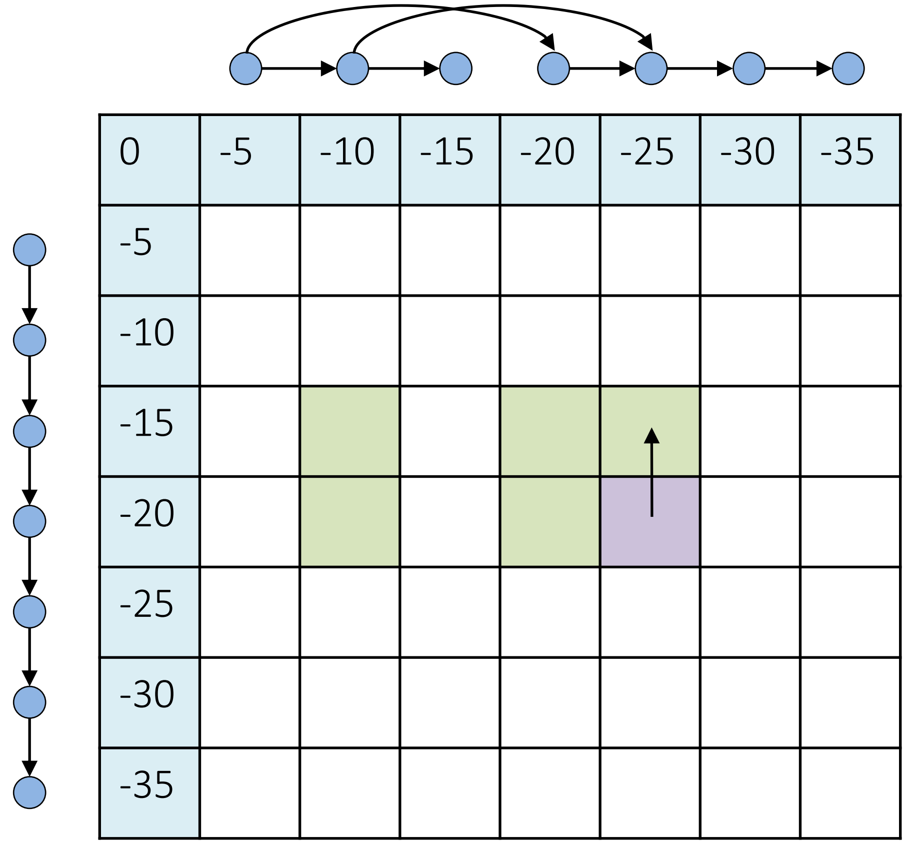
0	-5	-10	-15	-20	-25	-30	-35
-5							
-10							
-15							
-20							
-25							
-30							
-35							

🟪 Cell to score at current iteration

🟩 Scoring dependencies

← Dependency arc

Partial Order Alignment



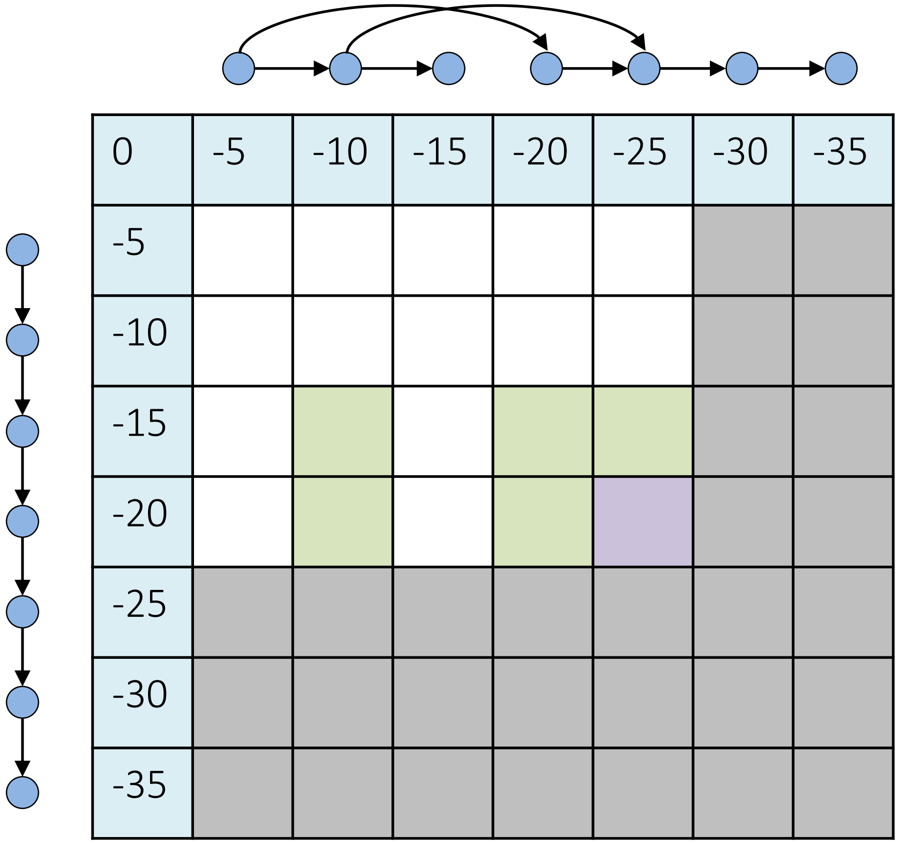
0	-5	-10	-15	-20	-25	-30	-35
-5							
-10							
-15							
-20							
-25							
-30							
-35							

🟪 Cell to score at current iteration

🟩 Scoring dependencies

← Dependency arc

Partial Order Alignment



0	-5	-10	-15	-20	-25	-30	-35
-5						▨	▨
-10						▨	▨
-15		🟩		🟩	🟩	▨	▨
-20		🟩		🟩	🟪	▨	▨
-25	▨	▨	▨	▨	▨	▨	▨
-30	▨	▨	▨	▨	▨	▨	▨
-35	▨	▨	▨	▨	▨	▨	▨

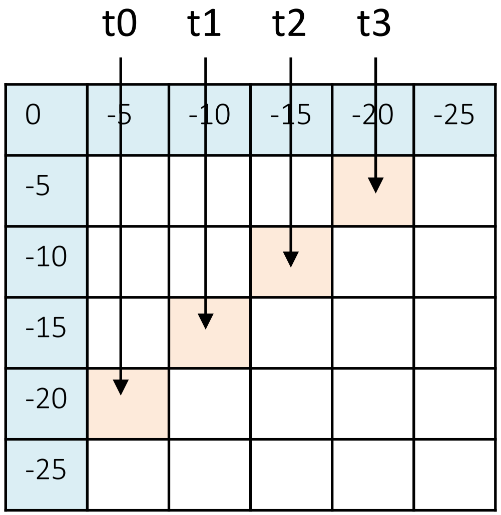
All the **white cells** are possible **scoring dependencies** for the current cell for a generic PO pair

🟪 Cell to score at current iteration

🟩 Scoring dependencies

← Dependency arc

PO alignment implementation



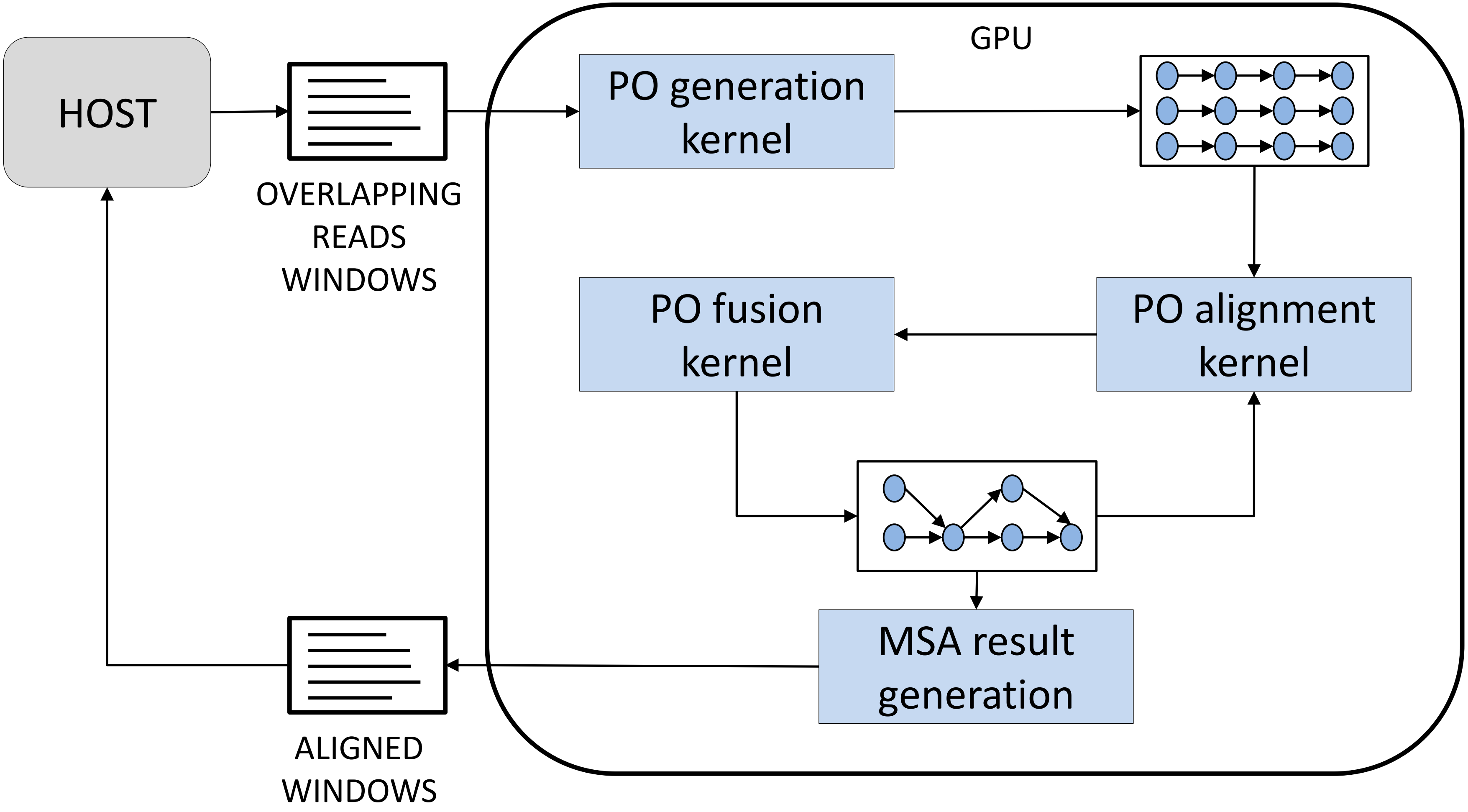
0	-5	-10	-15	-20	-25
-5					
-10					
-15					
-20					
-25					

- The PO graph is represented as and **edge list** stored in **shared memory**, plus a sequence of characters
- Each thread computes a cell of the current antidiagonal by looping over all the predecessors
- The scoring matrix is **stored by antidiagonals** for **coalesced memory access**

CHALLENGES

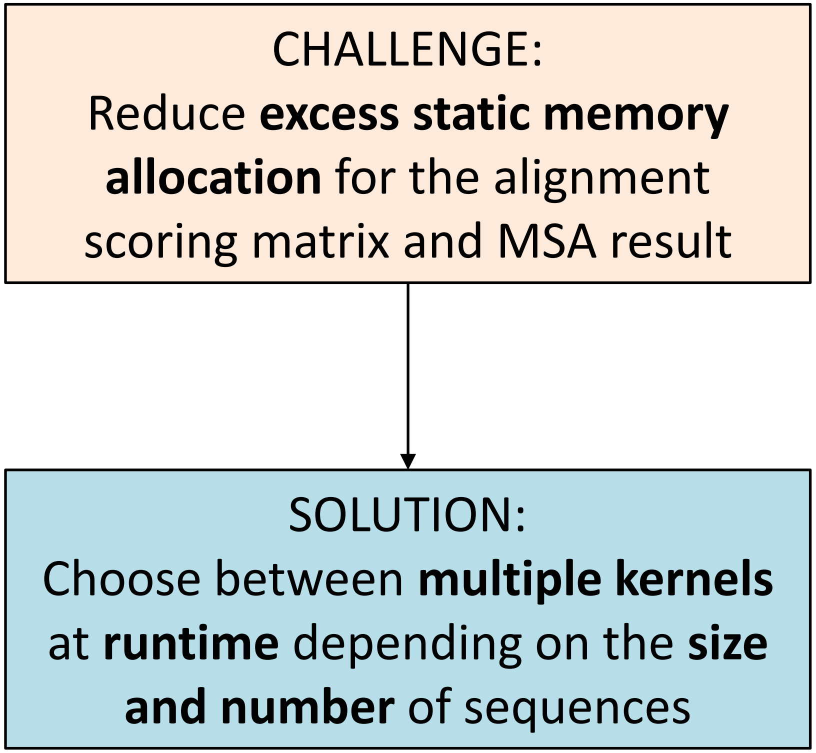
1. The dependencies of each cell change for different PO graphs, either pre-compute them or **store** the entire alignment matrix in memory (we chose the latter option)
2. The memory space required **changes** during the iterative alignment procedure -> allocate **enough memory statically** for each alignment

PO Multiple Sequence alignment



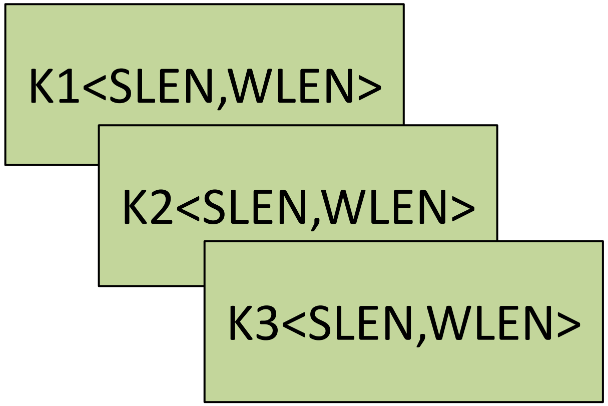
Each **CUDA block** operates on an **independent window of reads.** The whole **MSA task** is performed **in parallel** on up to 150,000 blocks

Kernel selection



CHALLENGE:
Reduce **excess static memory allocation** for the alignment scoring matrix and MSA result

SOLUTION:
Choose between **multiple kernels** at **runtime** depending on the **size and number** of sequences



K1<SLEN,WLEN>

K2<SLEN,WLEN>

K3<SLEN,WLEN>

Depending on the kernel selected and the device global memory capacity we can compute a **different number of blocks**

SLEN: maximum initial length of the sequences for each MSA task
WLEN: maximum number of sequences in the window for each MSA task

Roofline model analysis

- Given the specific nature of the parallelism in the alignment algorithm, we propose a theoretical ceiling in terms of **GWarpIntInsructions/s**:

$$IntF_{max} = \frac{1}{D}\sum_{k=1}^{D} \frac{F_{INT} \cdot N_k \cdot \mathrm{B}}{\lceil \mathrm{T} \cdot \mathrm{B}/\min(\mathrm{INT_C},\ \mathrm{T} \cdot \mathrm{SM} \cdot MB\ )\rceil}$$

$$\mathrm{T} = \left\lceil \frac{N_k}{T\ _s} \right\rceil \cdot \mathrm{T_s}$$

$\mathrm{INT_C} = number\ of\ integer\ FUs$

F_{INT} = frequency of an integer FU

SM = streaming multiprocessors

$\mathrm{T_s}$ = number of threads scheduled

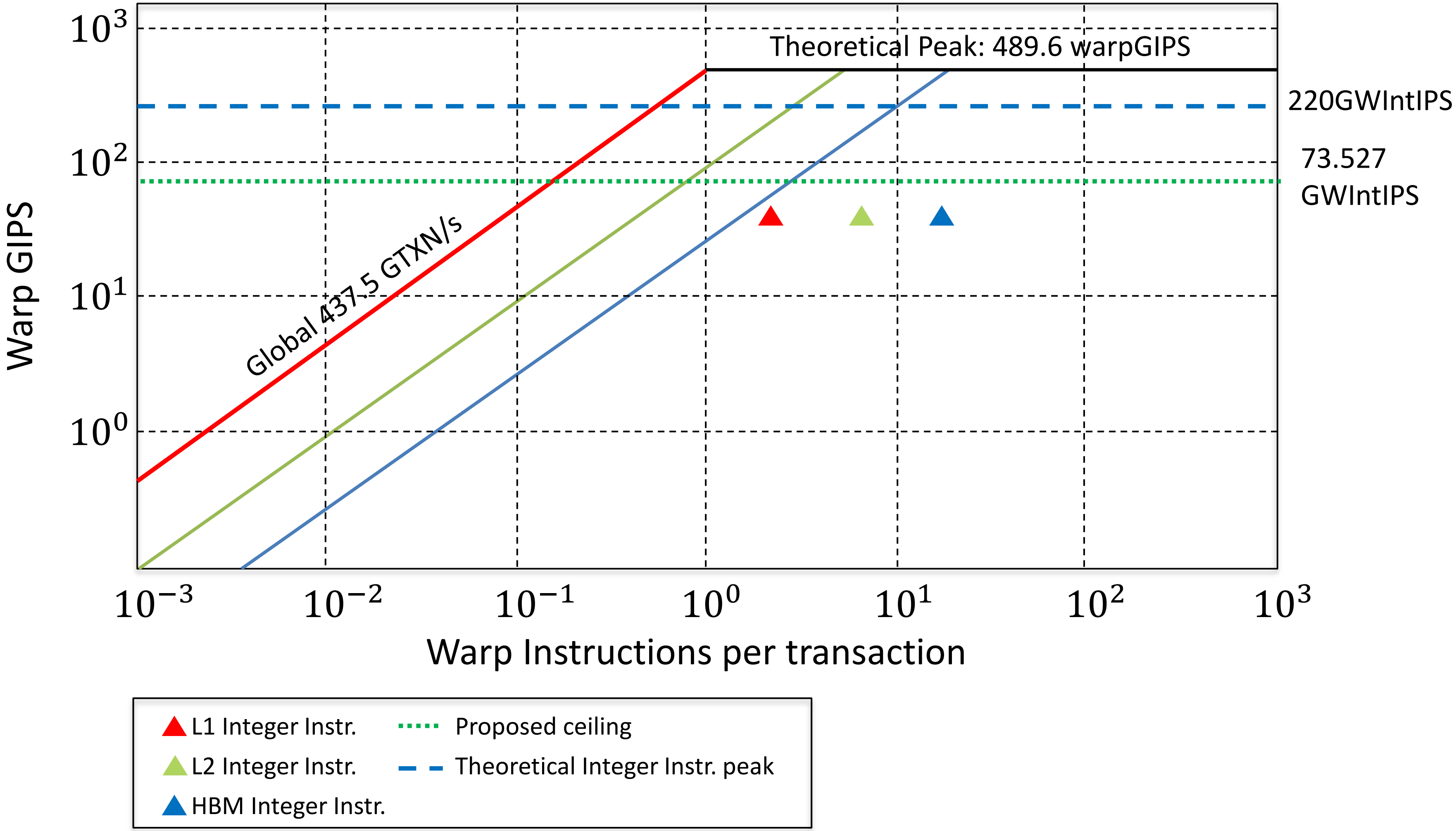
B = number of blocks scheduled

MB = max blocks per SM

$\mathrm{N_k}$ = elements to compute at iteration k

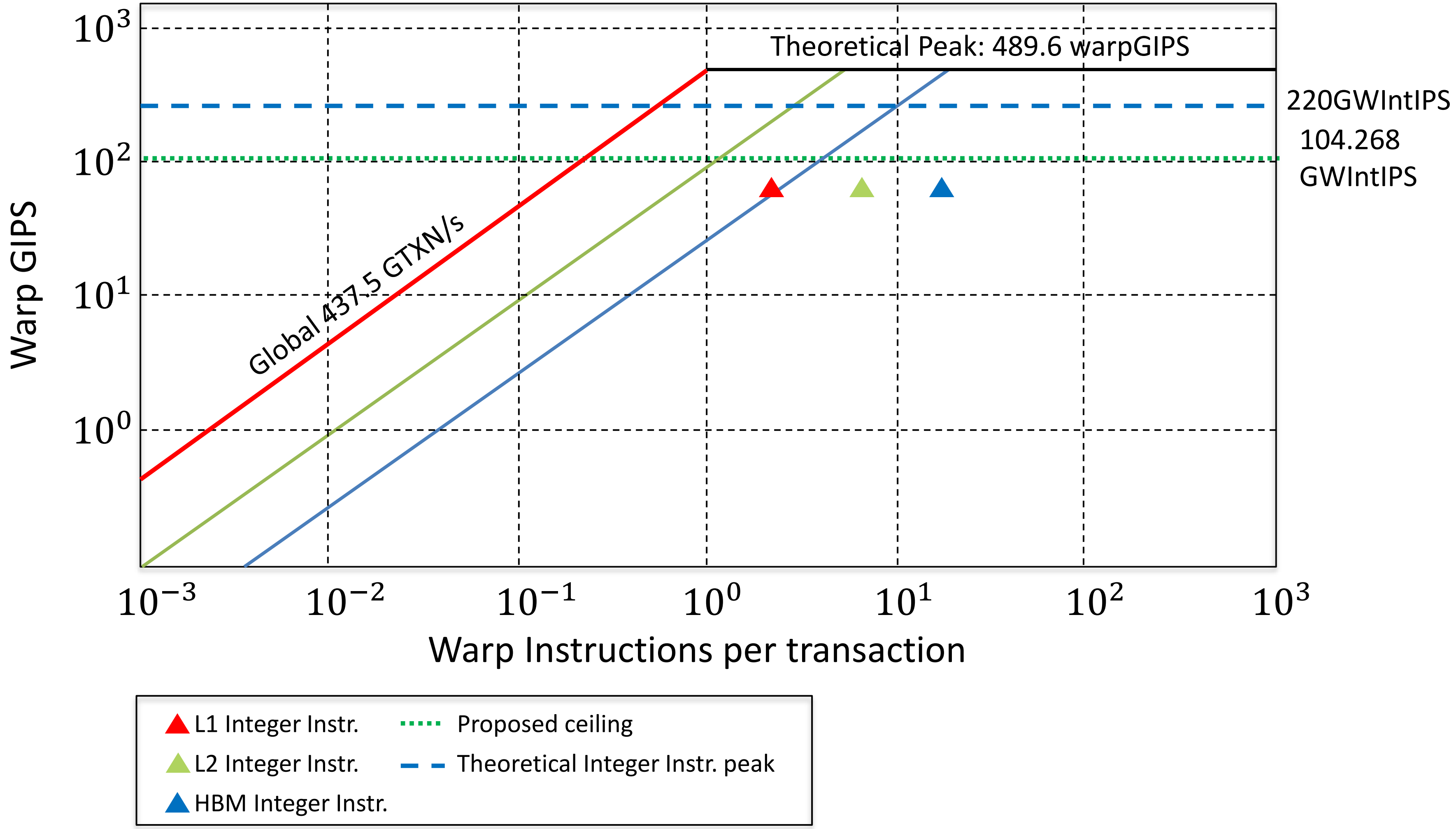
D = total iterations of the algorithm

Roofline model analysis



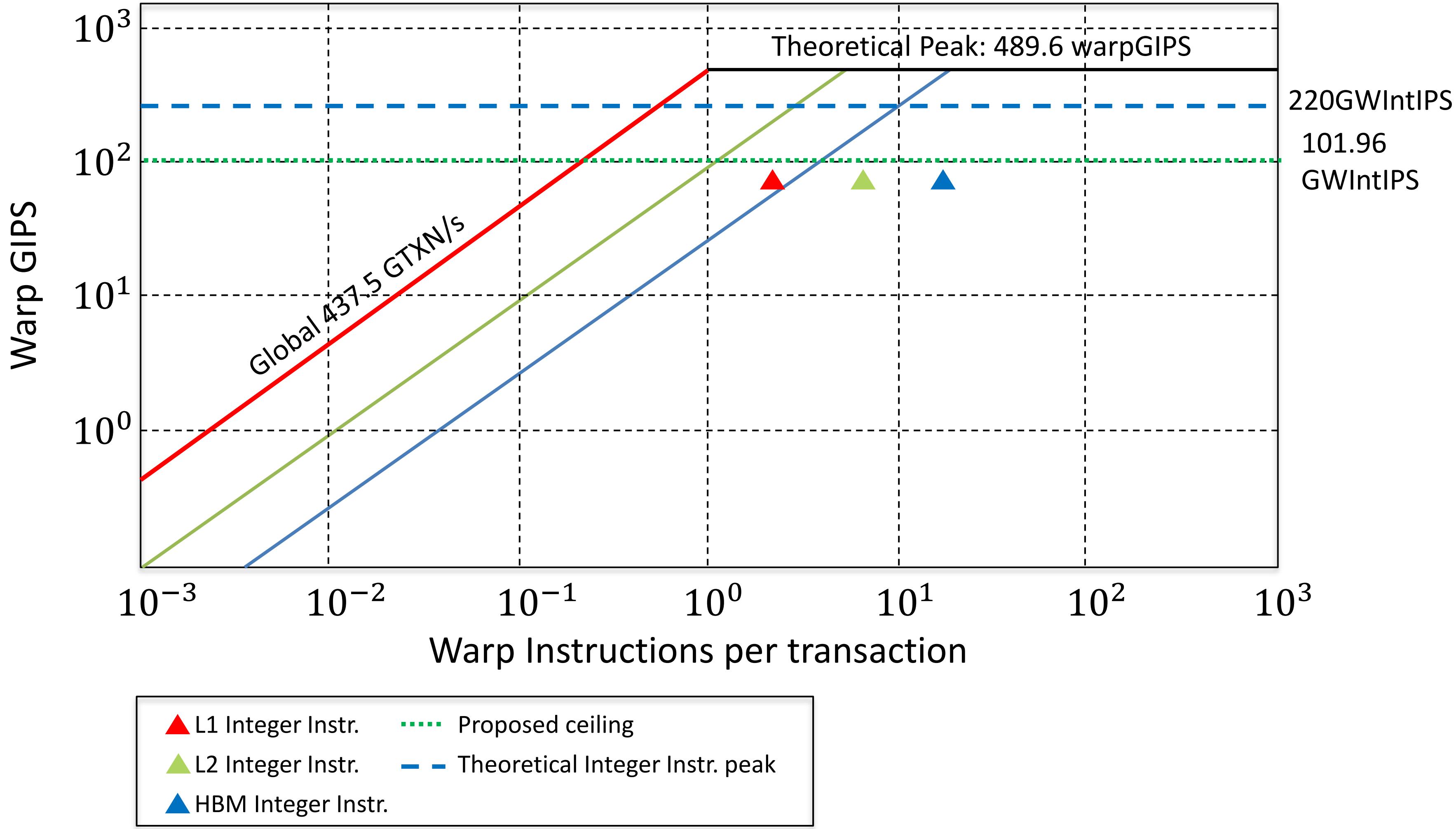
Roofline analysis for one GPU kernel for windows of between 1 and 32 sequences and sequences of 1-31 bp

Roofline model analysis



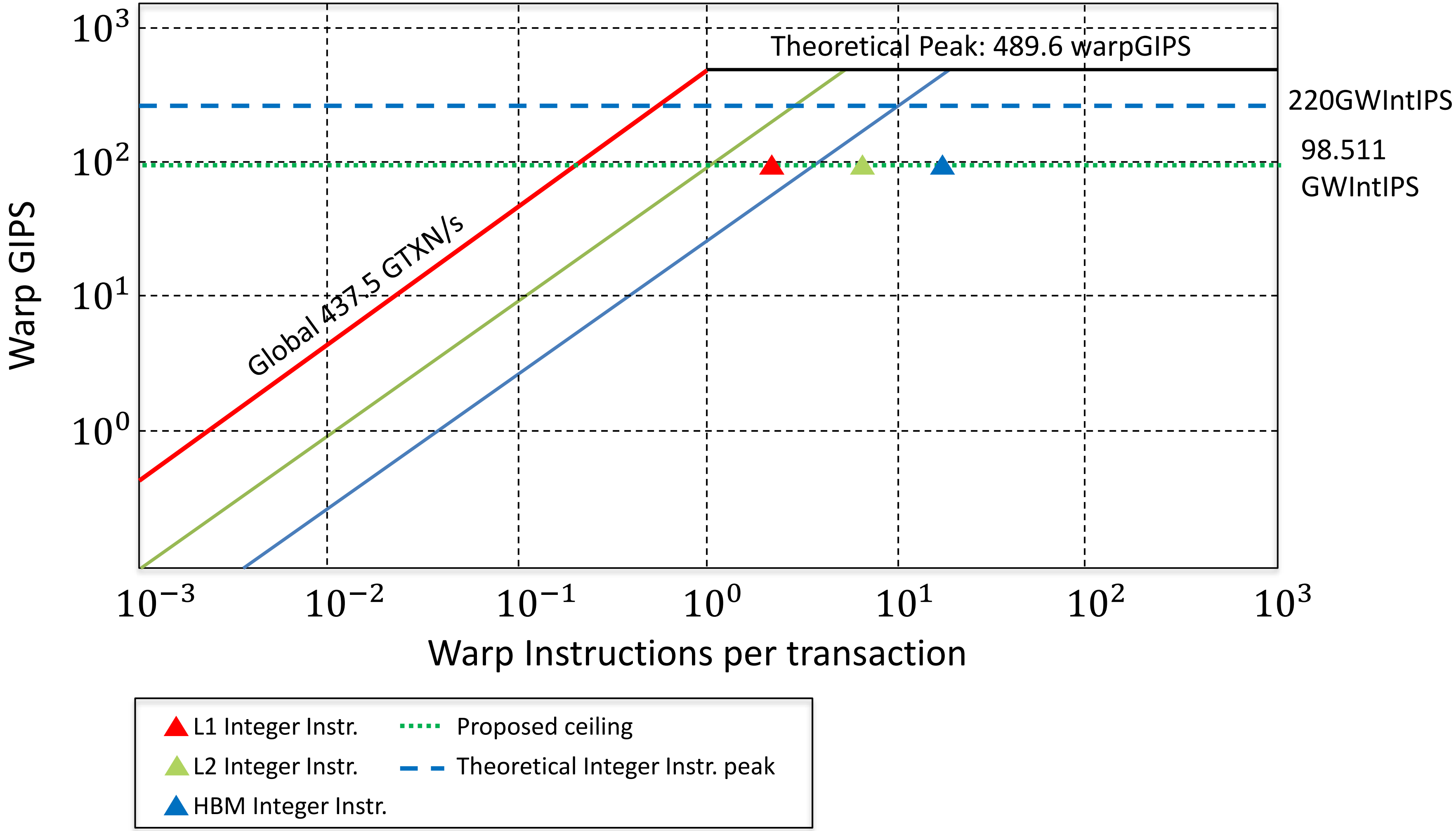
Roofline analysis for one GPU kernel for windows of between 1 and 32 sequences and sequences of 1-63 bp

Roofline model analysis



Roofline analysis for one GPU kernel for windows of between 1 and 32 sequences and sequences of 1-127 bp

Roofline model analysis



Roofline analysis for one GPU kernel for windows of between 1 and 32 sequences and sequences of 1-255 bp

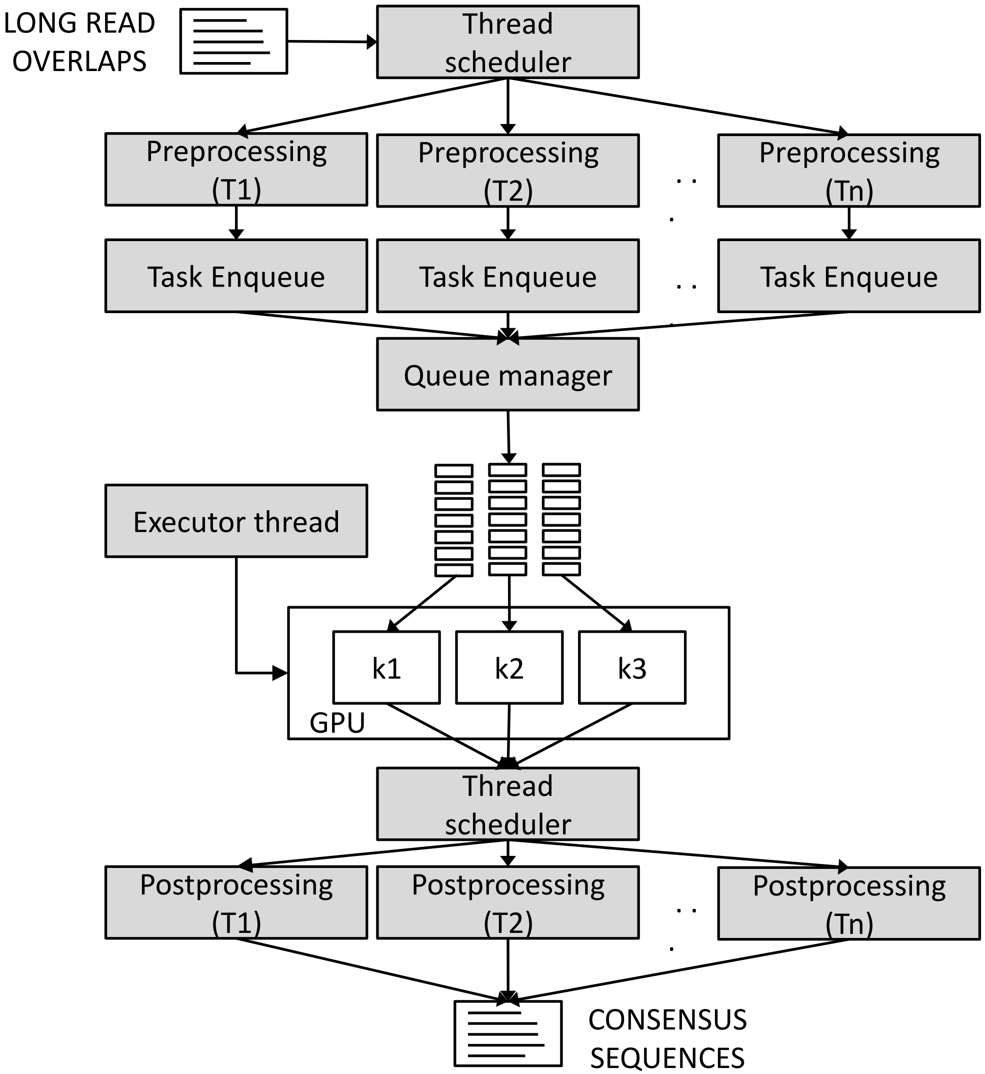
CONSENT integration

- The segmentation and correction strategy of CONSENT is split into three phases to create **batches of MSA tasks**

- Each thread is assigned to a **preprocessing and enqueue** task according to a round-robin policy

- The MSA tasks are enqueued in a **thread-safe queue**. Once the queue is full, the **executor thread** performs the accelerated MSA

- After the current batch of alignments has been performed, each thread is assigned to a **postprocessing** task to compute the final **consensus sequence** for the reads



Xeon E5 CPU performance comparison

Sequence size	Window size	CPU	Single thread speedup	64 threads speedup
1-32 bp	2-8	1 min 34s	35.31x	2.6x
32-63 bp	2-8	7 min 52s	82.15x	3.5x
64-127 bp	2-8	26 min 45s	121.28x	4.3x
128-255 bp	2-8	1h 42 min	192.13x	**6.49x**

Performance comparison of the PO alignment kernel executed on a **NVIDIA Tesla V100** against the CPU implementation of the BOA library [2] executed on a single thread and with 64 parallel threads on **two 2.3 GHz 16-core Intel Xeon Processors** E5-2698 v3 with a total of 64 hardware threads. Each experiment was executed on **1.2 million windows of sequences**.

Sequence size: number of base pairs for each individual sequence in the MSA

Window size: number of sequences in the MSA procedure

[2] https://github.com/Malfoy/BOA

Skylake CPU performance comparison

Sequence Size	Window size	CPU	GPU	Speedup
1-32 bp	17-32	2 min 25s	1 min 7s	2.16x
32-63 bp	17-32	4 min 45s	1 min 57s	2.43x
64-127 bp	17-32	11 min 29s	4 min 19s	2.65x
128-255 bp	17-32	36 min 44s	12 min 55s	**2.84x**

Performance comparison of the PO alignment kernel against the CPU implementation of the BOA library executed with 80 parallel threads on **two Intel Xeon Gold 6148 ('Skylake')** running at 2.40 GHz. Both were executed on **3.2 million windows of sequences.**

Sequence size: number of base pairs for each individual sequence in the MSA

Window size: number of sequences in the MSA procedure

*A more complete version of this table is available in the paper

GPU state of the art comparison

Sequence size	Window size	Our kernel	Clara Genomics[1]	Speedup
1-255 bp	1-32	2 min 40s	15 min 42s	**5.89x**

[1] Clara Genomics run on single CUDA stream in MSA generation mode (same type of output as our implementation)

Sequence size	Window size	Our kernel	Clara Genomics[2]	Speedup
1-255 bp	1-32	2 min 40s	10 min 28s	**3.92x**

[2] Clara Genomics multi-batch benchmark in consensus generation mode (different type of output, but the most efficient way to run Clara Genomics, included for completeness)

All experiments are performed on a **NVIDIA Tesla V100** on **2 million windows** of sequences

CONSENT acceleration results

Dataset	Organism	Dataset size	CONSENT-GPU	CONSENT	Speedup
SRR10326407	E. Coli(30x)	151 Mbp	6 min 29s	34 min 36s	5.3x
SRR10326407	E. Coli(60x)	290 Mbp	16 min 44s	2h 26 min	**8.5x**
SRR7743079	D. Melanogaster (20x)	2.9 Gbp	2h 53 min	6h 17 min	2.18x
ERR3454401	S. Cerevisiae (30x)	386 Mbp	1h 6 min	23 min	2.86x
ERR3454401	S. Cerevisiae (60x)	756 Mbp	3h 0 min	1h 32 min	1.95x

Performance comparison of CONSENT and our GPU accelerated version. Both software were run on **two Intel Xeon Gold 6148 ('Skylake')** running at 2.40 GHz with **80 parallel threads**.

Conclusions

- We presented a **GPU accelerated** algorithm for **multiple sequence alignment** based on **partial order graphs** that outperforms the state-of-the-art CPU-based POA v2 alignment library for the targeted range of sequence and window lengths, achieving a speedup that ranges from **2.16x** to **6.49x**

- To evaluate the quality of the proposed GPU implementation, we have devised an extension of the **Roofline model for GPU** and we show that our kernel achieves near-optimal performance

- We have also shown that for the target range of sequences and window lengths we outperform the Clara Genomics PO alignment module by **5.89x** and **3.92x** for different execution modes on the NVIDIA Tesla V100 GPU

Contacts

For questions regarding this work, email
Francesco Peverelli: francesco1.peverelli@mail.polimi.it

Github repository:
https://github.com/francesco-peverelli/CONSENT-GPU