Jonathan Lévy
Master Thesis Defence

Acceleration of Seed Extension for BWA-MEM DNA Alignment Using GPUs

1) Introduction

- Cancer detection with DNA analysis
- Time-consuming: needs acceleration
- Speed-up with parallel computing

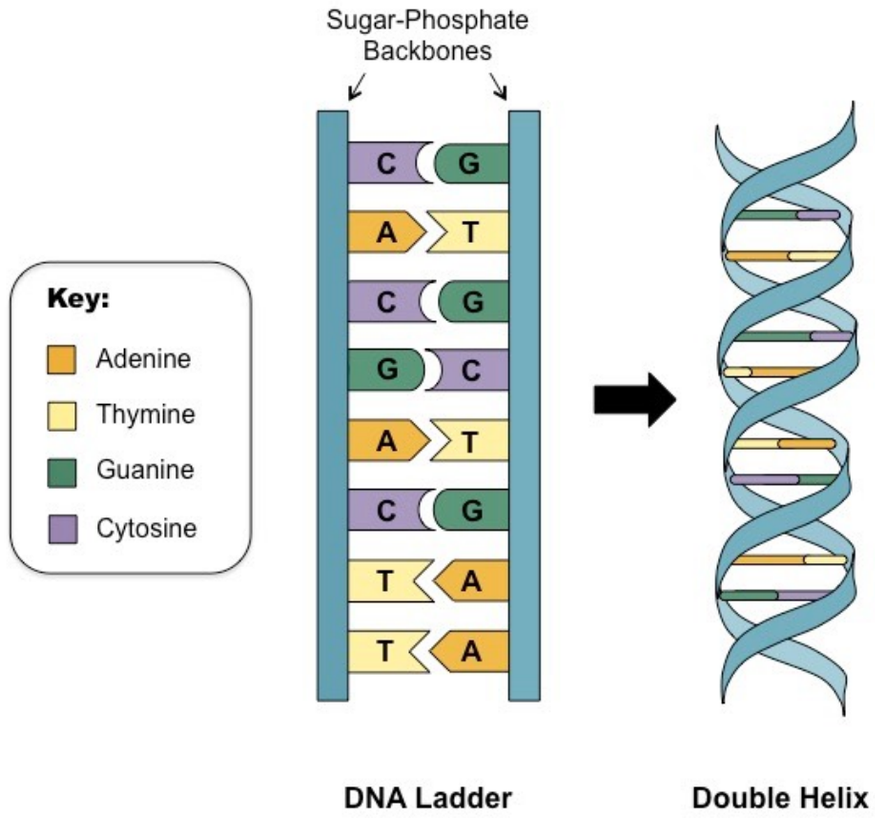
- Goals:
  - Speed
  - Adaptability
  - Maintainability
  - Correctness

Plan

1) Introduction: DNA, mapping, GPU
2) Accelerator design
3) Implementation
4) Results

1) Introduction: DNA

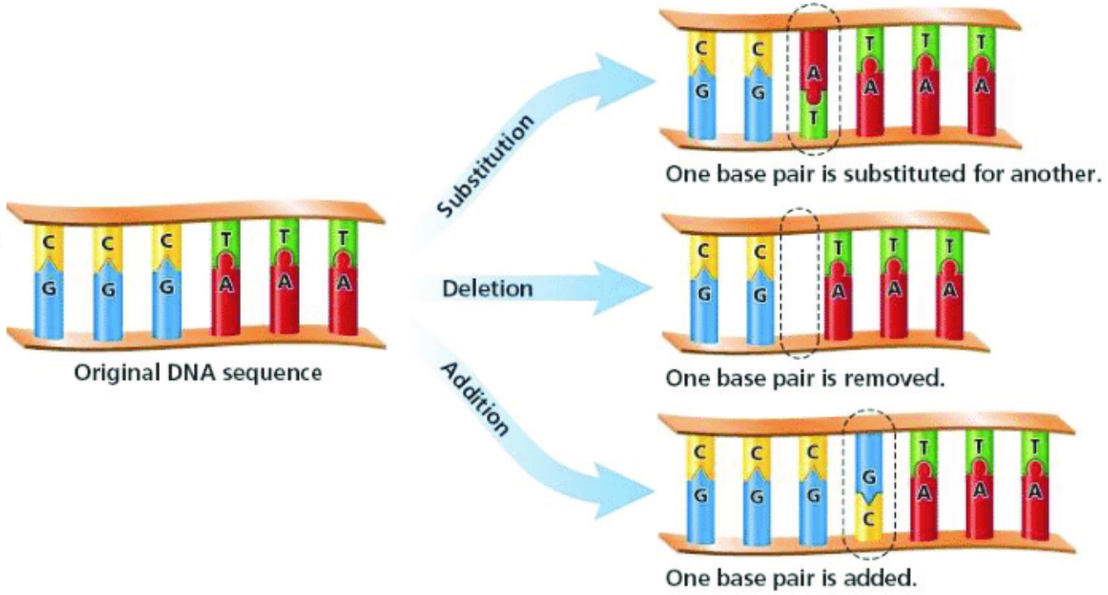
- DNA: code that defines species
- 4 bases: Adenine, Thymine, Cytosine, Guanine
- Paired 2 by 2:
  - A – T
  - C – G
- Double-stranded helix



1) Introduction: DNA mapping

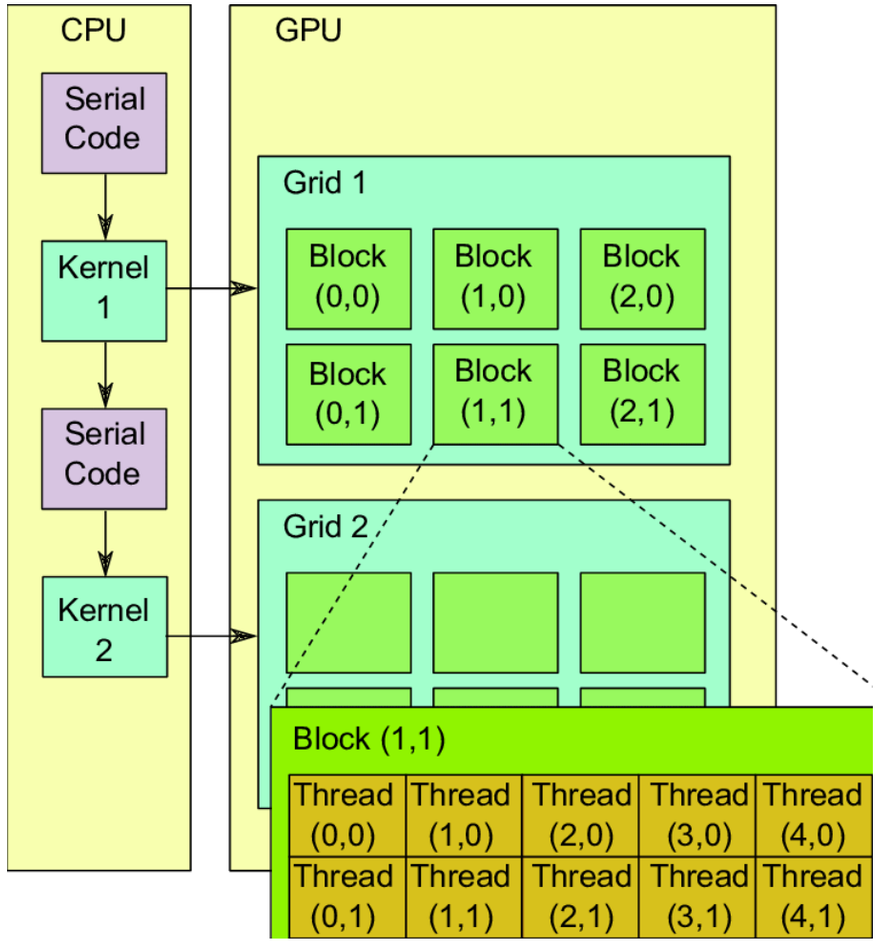
Mutation: DNA modification, can be harmful

1) Get DNA sample
   - Sample: set of fragments “reads”
2) Find matches in a *reference genome*
   - Find location
   - Quantify how well it matches



1) Introduction: parallel processing

- Millions of strings, unrelated
- Parallel processing on GPU
  - Graphics Processing Units
  - Parallel hardware
  - Can be programmed for any kind of computation
- CPU launches *kernels* on GPU



1) Introduction: seed-and-extend

Mapping a read with *seed-and-extend*:

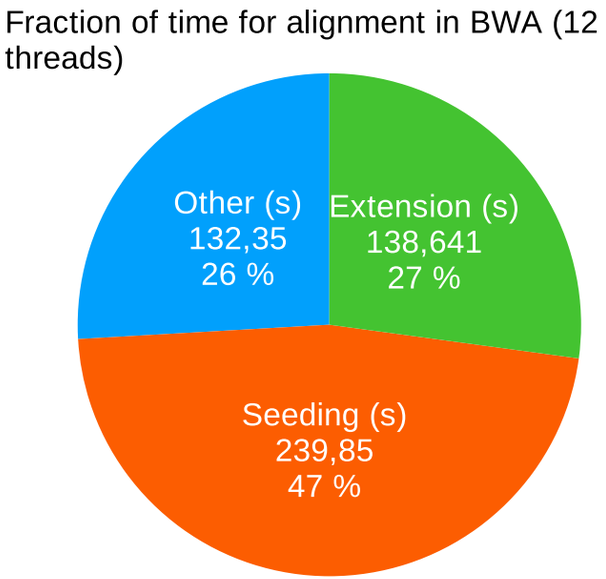
1) Find matching area – *seeding*

   Find small (~20 bases) substrings in reference genome

2) Align rest of the read – *seed-extension (~30% of time)*

   1) For each substring, align left and right parts

   2) Compute dynamic programming matrix (Smith-Waterman algorithm)



2) Accelerator design: programs

BWA-MEM

- DNA mapper
- Uses seed-and-extend
- CPU threads (12~24)

GASAL2

- CUDA library, aligns DNA
  - Runs the seed-extension
- Smith-Waterman & others
- Compresses data on GPU
- GPU threads (2000~3000)

Integrate GASAL2 in BWA-MEM

2) Accelerator design: task

- Local alignment:
  - Compute matrix of scores
  - Bases match → score
  - Bases mismatch → score
  - Maximum score = end position of the alignment
- Start with non-zero score
- Affine gap penalty

S		g	g	g	c	t	g	g	c	g	a
	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	2
g	0	2	2	2	1	0	2	2	1	2	1
g	0	2	4	4	3	2	2	4	3	3	2
c	0	1	3	3	6	5	4	3	6	5	4
g	0	2	3	5	5	5	7	6	5	8	7
g	0	2	4	4	4	4	7	6	5	7	7

2) Accelerator design: task

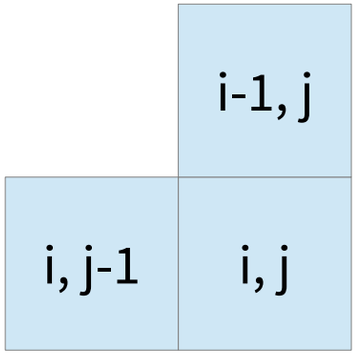
Affine gap penalty

- α > 0 : bases match
- g < 0 : cost to open a gap
- h < 0 : cost to widen a gap

$$S[i,j] = max \begin{cases} S[i-1,j-1] + \alpha & \text{if} \quad a_i = b_j \\ G_A[i,j] \\ G_B[i,j] \end{cases}$$

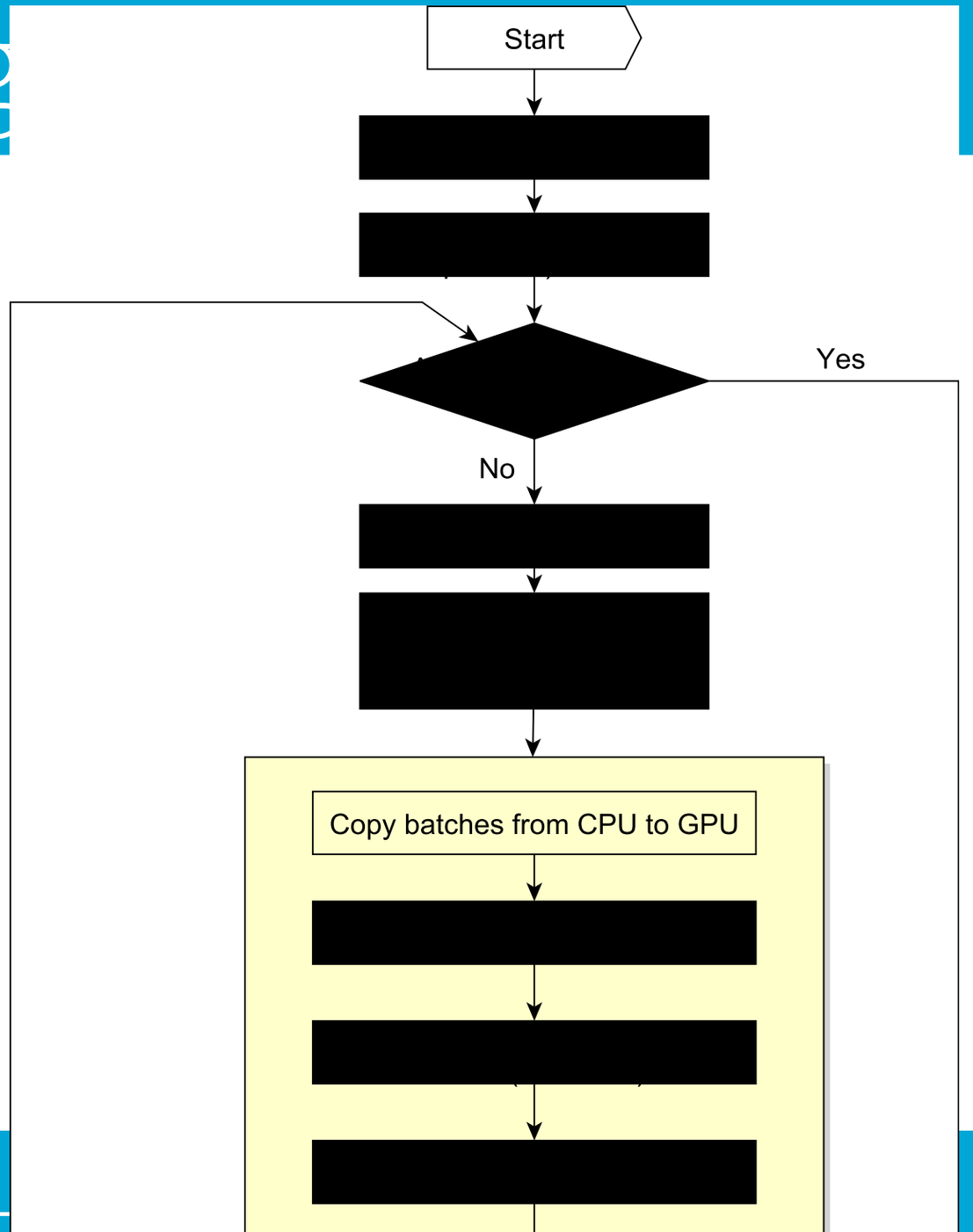
$$G_A[i,j] = max \begin{cases} S[i-1,j] + g + h \\ G_A[i-1,j] + h \end{cases}$$

$$G_B[i,j] = max \begin{cases} S[i,j-1] + g + h \\ G_B[i,j-1] + h \end{cases}$$



2) Accelerator desig

- GASAL2 Data flow
  - Initialise structures
  - Fill an available stream
  - Launch kernel for stream
  - Retrieve any finished stream
  - Repeat for all batches of sequences



2) Accelerator design: specifications

- Reproduce original BWA-MEM behaviour
- Have dynamic memory allocation
- Compute 2 sides in parallel
- Provide effective speed-up
- Provide minimum difference in output

Measures:

- Execution time (s)
- Output difference (%)
- Resources use:
  - Memory (MB)
  - SM utilisation (%)

3.1) Implementation: maintainability

- C++ templates
- Values as types
  - Resolution by compiler
- Reduces code amount
- Generates specialised kernels

  → no branch resolution at runtime

```
1  template <int Val>
2  struct Int2Type{
3          enum {val_ = Val} dummy;
4  };
5  template<typename X, typename Y>
6  struct SameType {
7          enum { result = 0 };
8  };
9  template<typename T>
10 struct SameType<T, T> {
11         enum { result = 1 };
12 };
13 #define SAMETYPE(a, b) (SameType<a,b>::result)
```

3.1) Implementation: C++ port

- GASAL2: needs to process batches
- Modified BWA-MEM with batches: GASE-GASAL2
- Ported GASE-GASAL2 to C++
- Integrated new GASAL2

3.2) Implementation: kernel

- Reproduce BWA-MEM behaviour “ksw_extend”
- Custom Smith-Waterman
  - Non-zero start score
  - Cell skipping
  - Z-dropoff

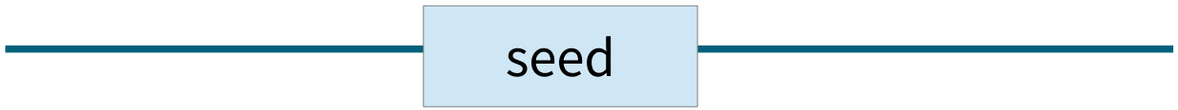
1) Simple kernel implementation
2) Added speed-up cell-skipping
3) Added z-droppoff
4) Compared: outputs same results

3.3) Implementation: library integration

- Seed filtering
  - Heuristics in GASE-GASAL2 (85% extension)
- Left-right dependency
  - Left starts with seed score
  - Right starts with left score

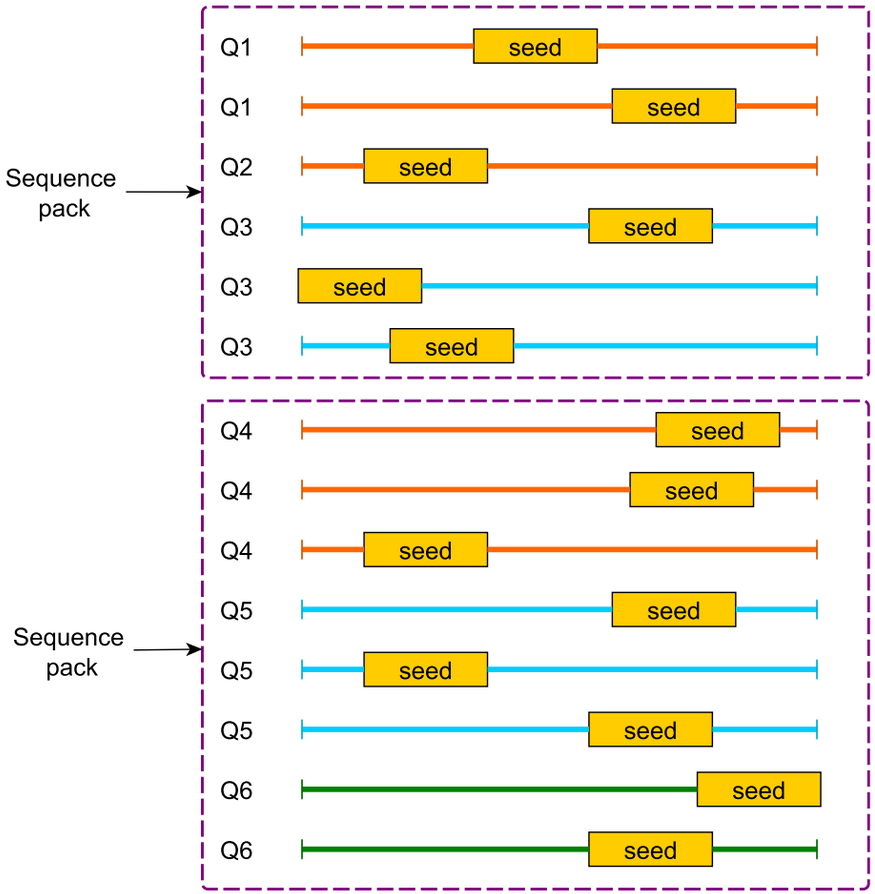
Introducing: “seed-only” paradigm

- Start score = seed score
- Both sides in parallel
- Could bring minor differences



3.3) Implementation: library integration

- Compute left & right in parallel
  - 2 batches
  - Grouping by long/short instead of left/right
  - More instructions executed
- CPU thread: packs of 4000
  - GPU stream: batch of 1000
  - Process 1000 seq.
  - Generates 100 000 seeds
  - Generates 180 000 alignments
  - Split in 2 batches depending on length



- Sequence pack = 4000 seq.
- Same colour = same GPU stream
- Ex. with 2 streams:
  - Orange: Stream #1
  - Blue: Stream #2
  - Green: Stream #1

3.4) Implementation: memory

- GASAL2: max memory at compilation
- Unacceptable
  - Unknown number of seeds
  - Unknown length of parts to align
- Dynamic memory

- 2 different memory schemes
  - Simple CudaRealloc for metada fields
  - Extensible structure for DNA strings

3.4) Implementation: memory

Metadata:

- Implemented a CudaRealloc
  - CUDA fields: no “realloc”
  - Creates bigger field
  - copy content
  - free previous
- User checks if overflow, call resizer

DNA strings:

- Linked-list structure
  - Grows bigger nodes
  - No memory copies
  - No in-between frees
  - Fast & efficient
- Library concern
- Interface for user

3.5) Score comparison

- Diff UNIX utility
- Difference percentage

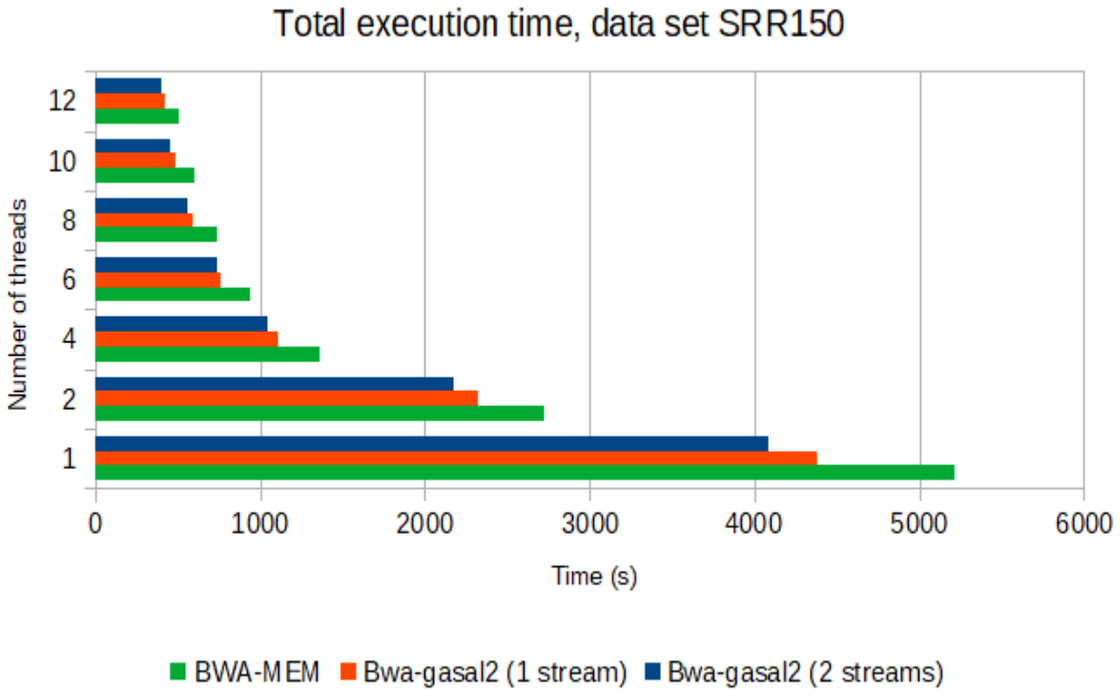
- Remove secondary scores
- Enormous files (~GB), sed too slow
- Wrote C program for files processing

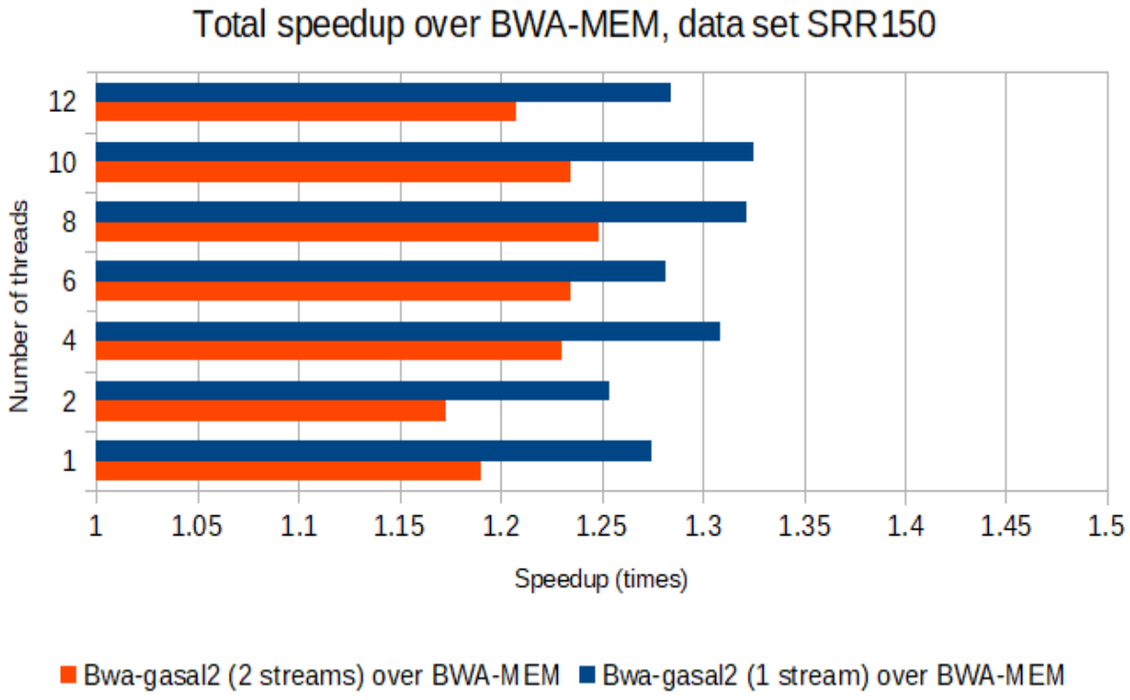
4.1) Results

Test machine specifications

Component	Reference
CPU	2x Intel Xeon E3-2620 (Total 12 physical cores, 24 logical cores)
RAM	32GB DDR3
GPU	NVIDIA Tesla K40c (2880 CUDA cores, VRAM 12GB GDDR5)
Operating System	GNU/Linux Red Hat Enterprise Linux 7
CUDA version	10.1.105
GCC version	4.8

4.2) Results: speed-up

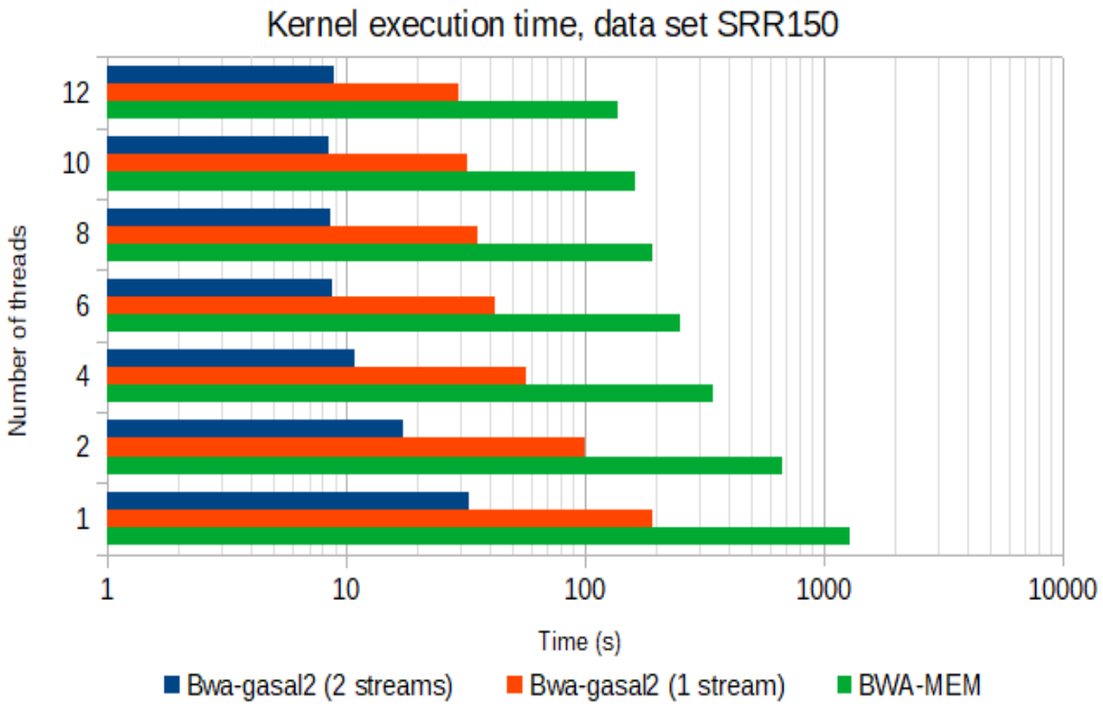


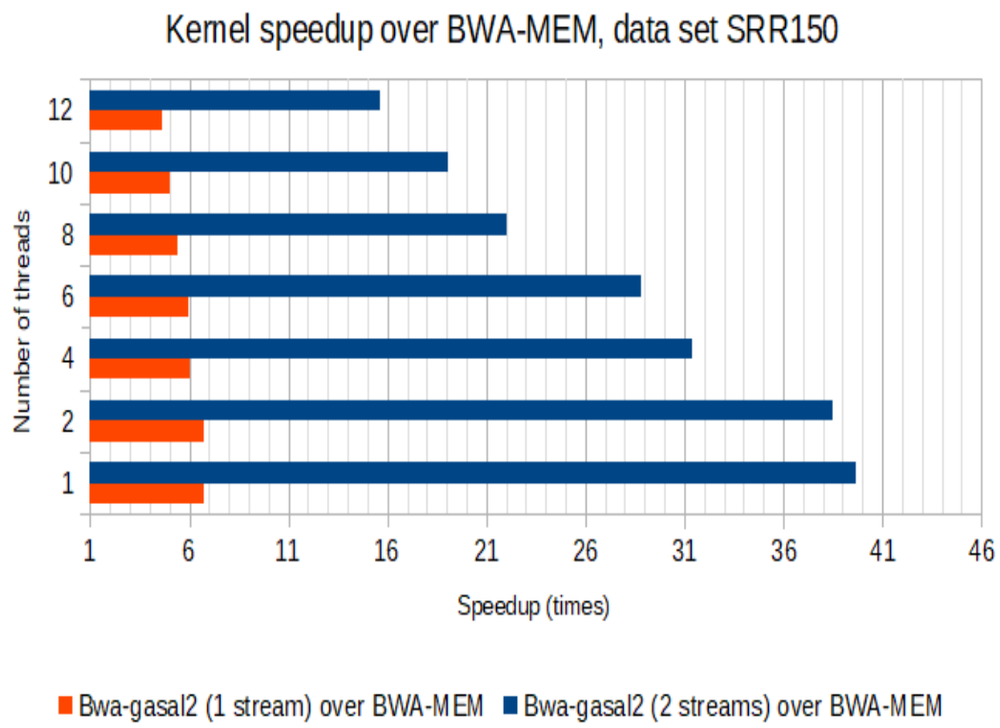


12 threads kernel speed-up:

- 1.21x with a single stream
- 1.28x with 2-streams (1.37x theoretical max)

4.2) Results: kernel speed-up



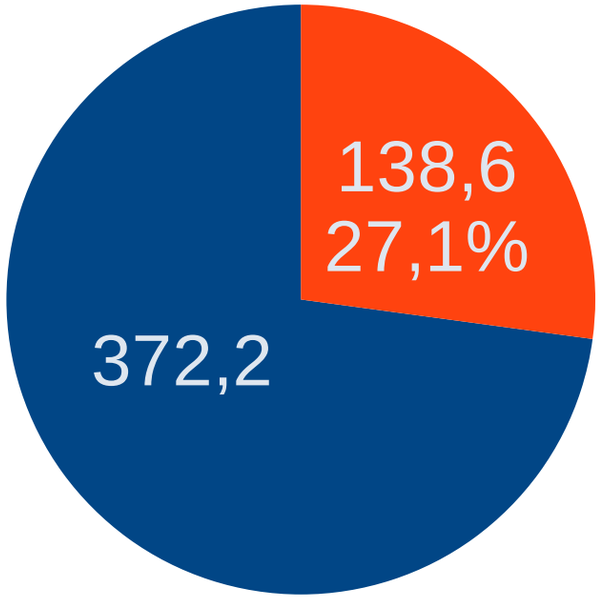


12 threads kernel speed-up:

- 4.8x with a single stream
- 16x with 2-streams

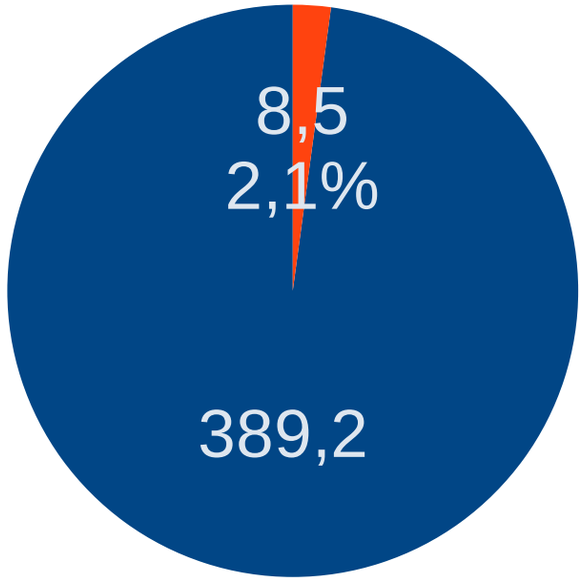
4.2) Results: speed-up

Fraction of time for alignment in BWA (12 threads)



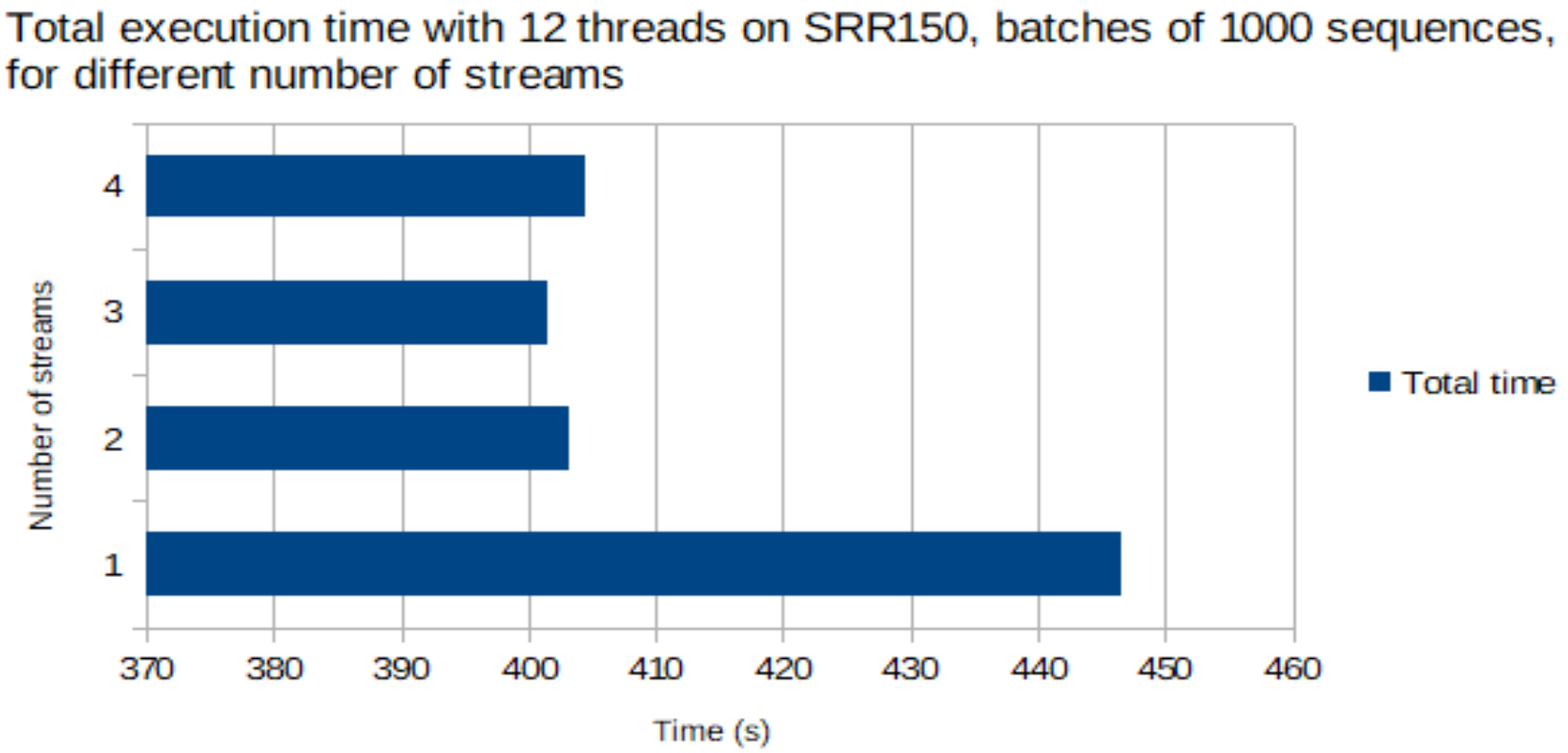
Rest (s) Alignment (s)

Fraction of time for alignment in BWA-GASAL2 (with hidden time, 12 threads)



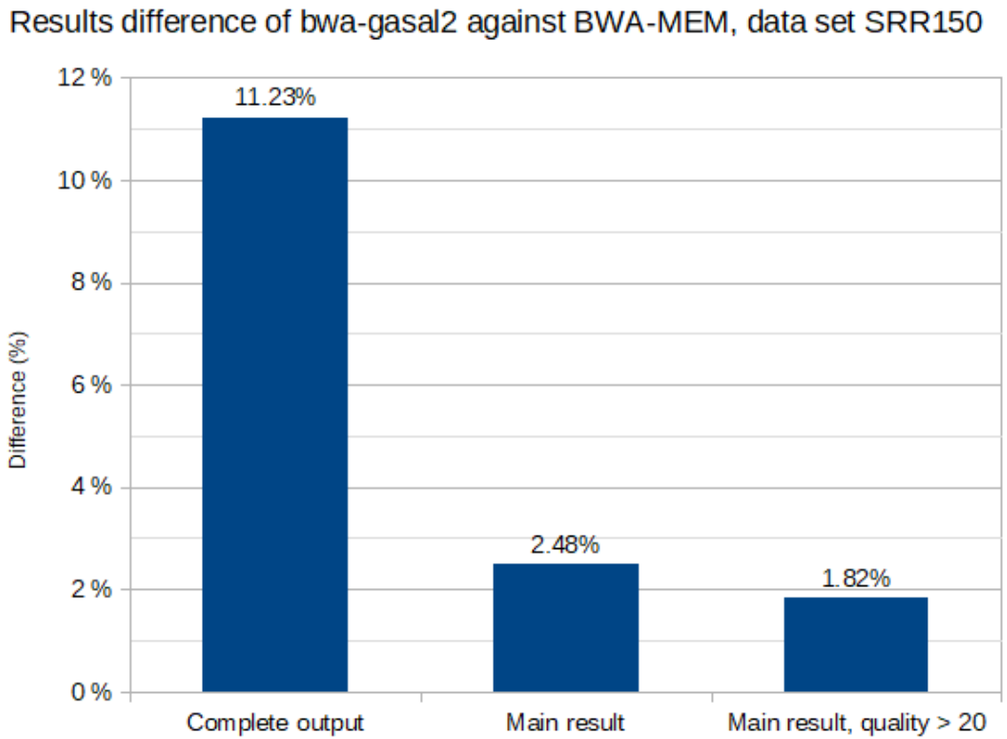
Rest (s) Alignment (s)

4.2) Results: streams and overlap



No speed-up after 2 streams → sufficient for CPU-GPU overlapped execution

4.3) Results: Correctness



- Total differences: 11.23%
- Main score only: 2.48%
- Main score, good quality alignments: 1.82%

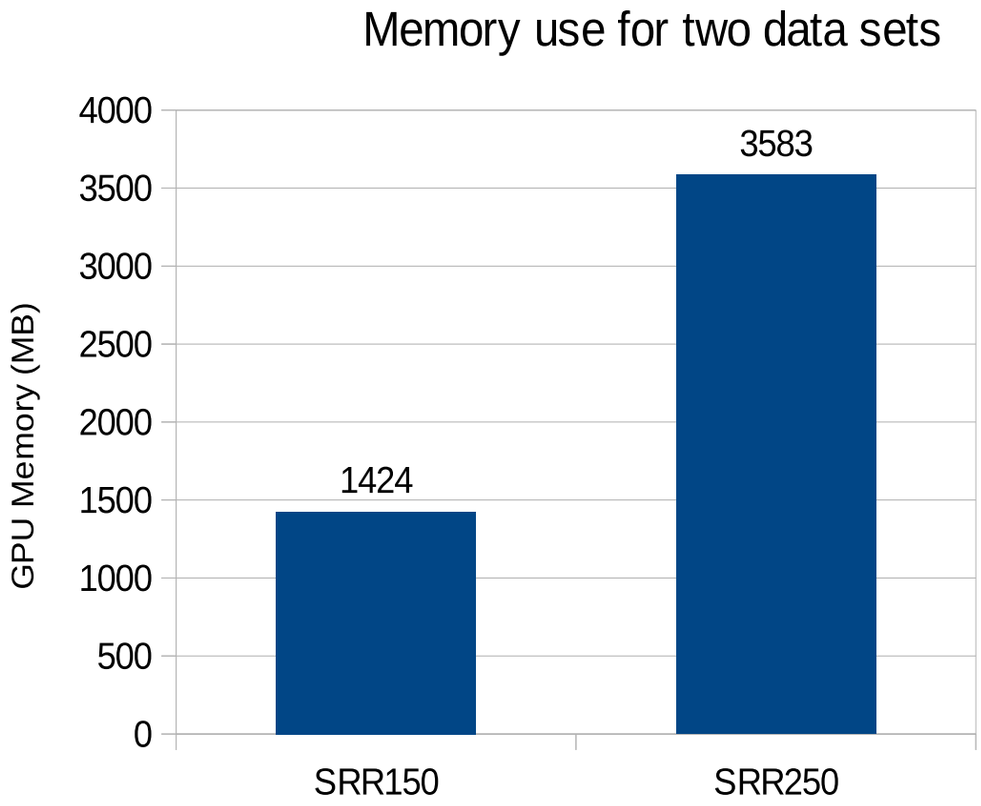
  → few differences

  → result is acceptable

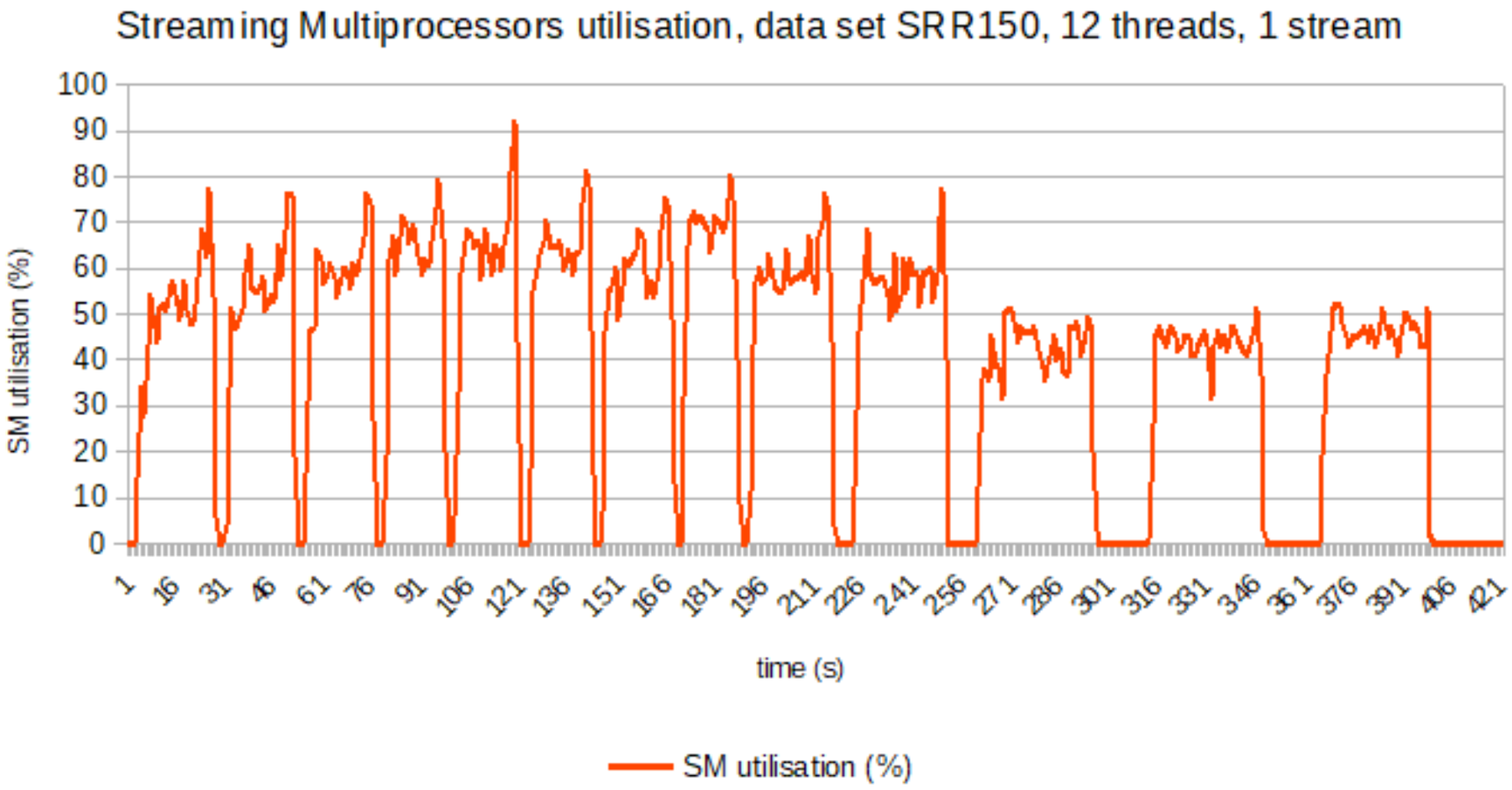
4.4) Results: resources use

Dynamic allocation

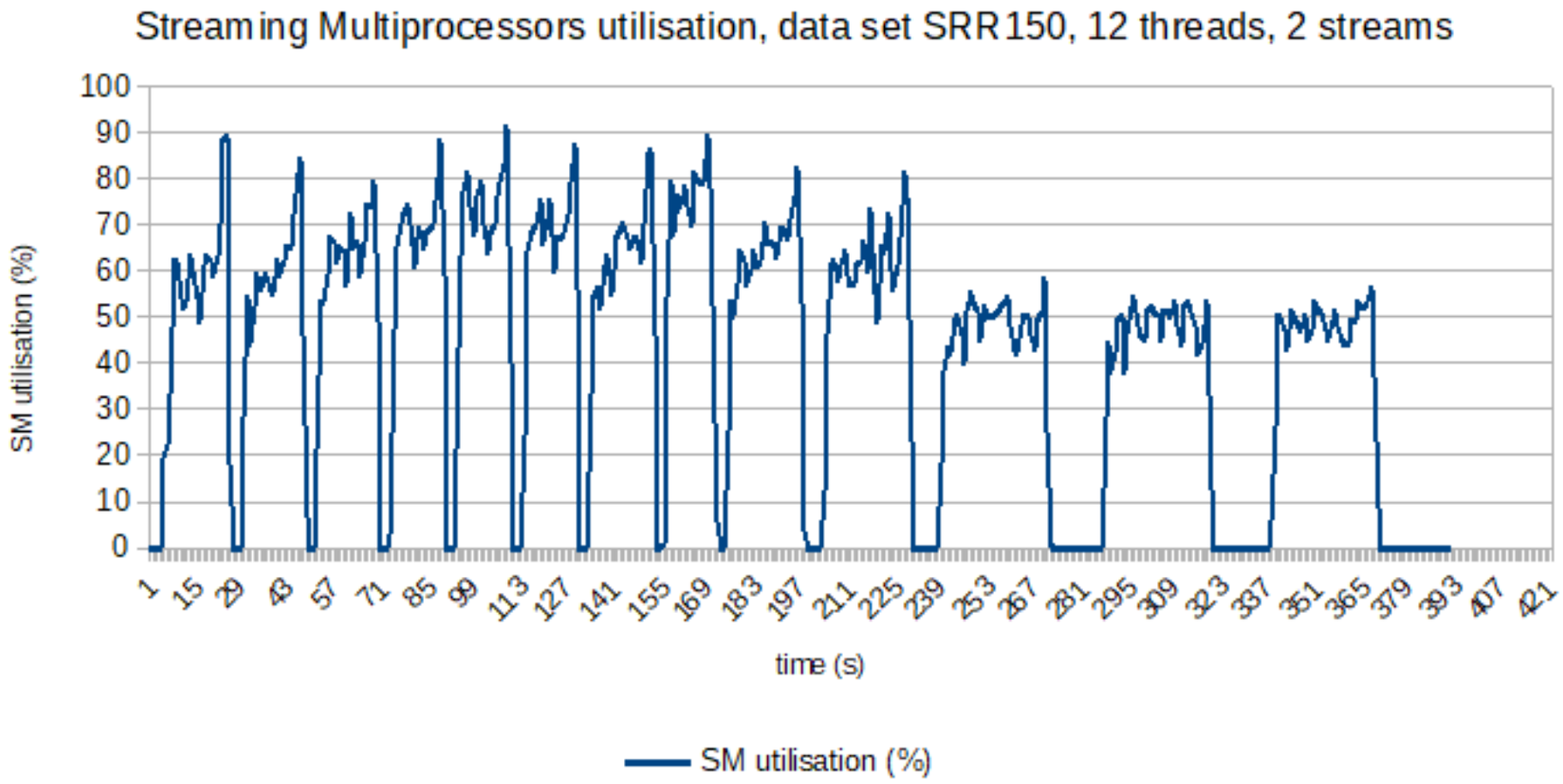
- Reasonable memory use
- Room to accelerate another part of calculation
- Lower-end hardware use is possible



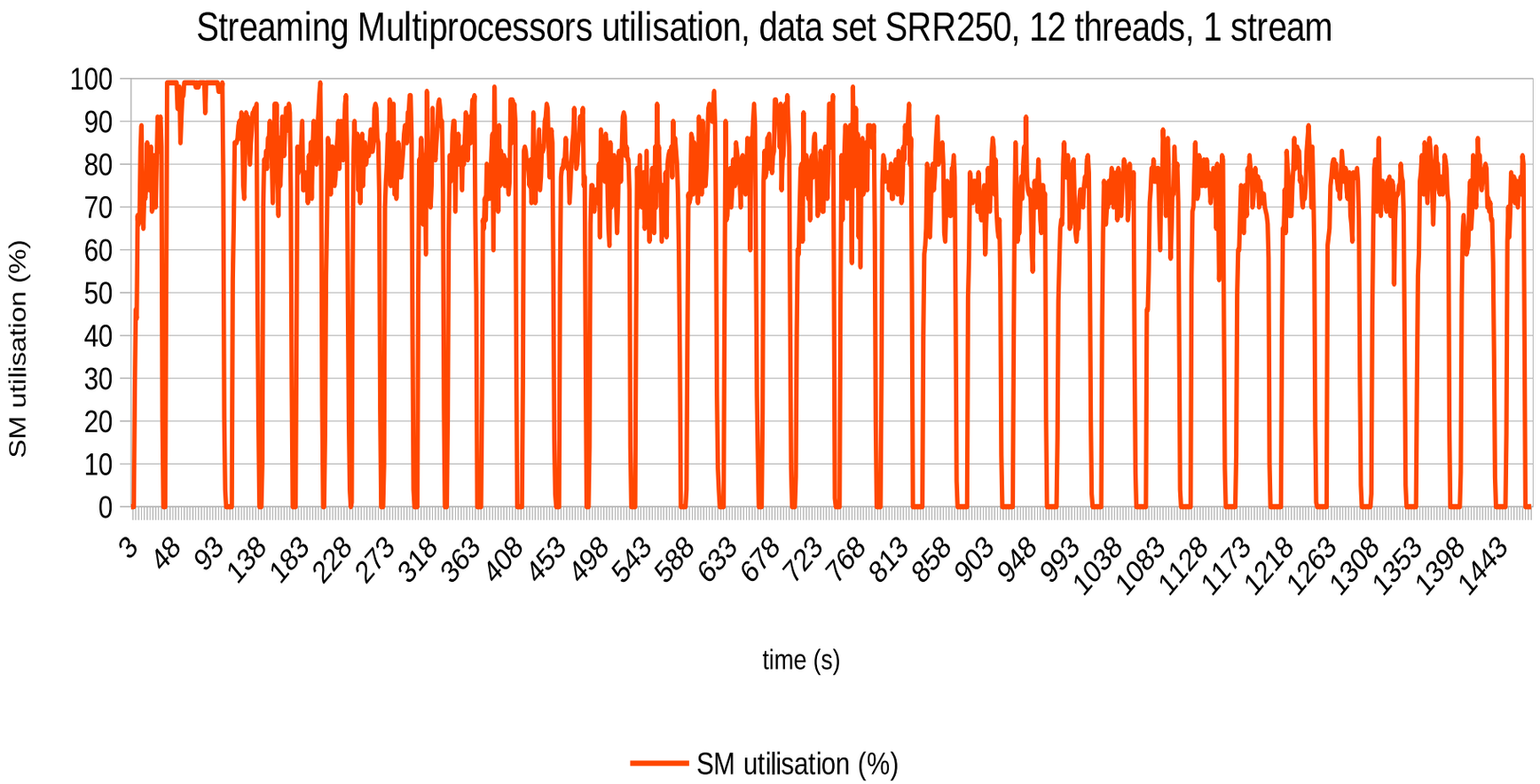
4.4) Results: resources use



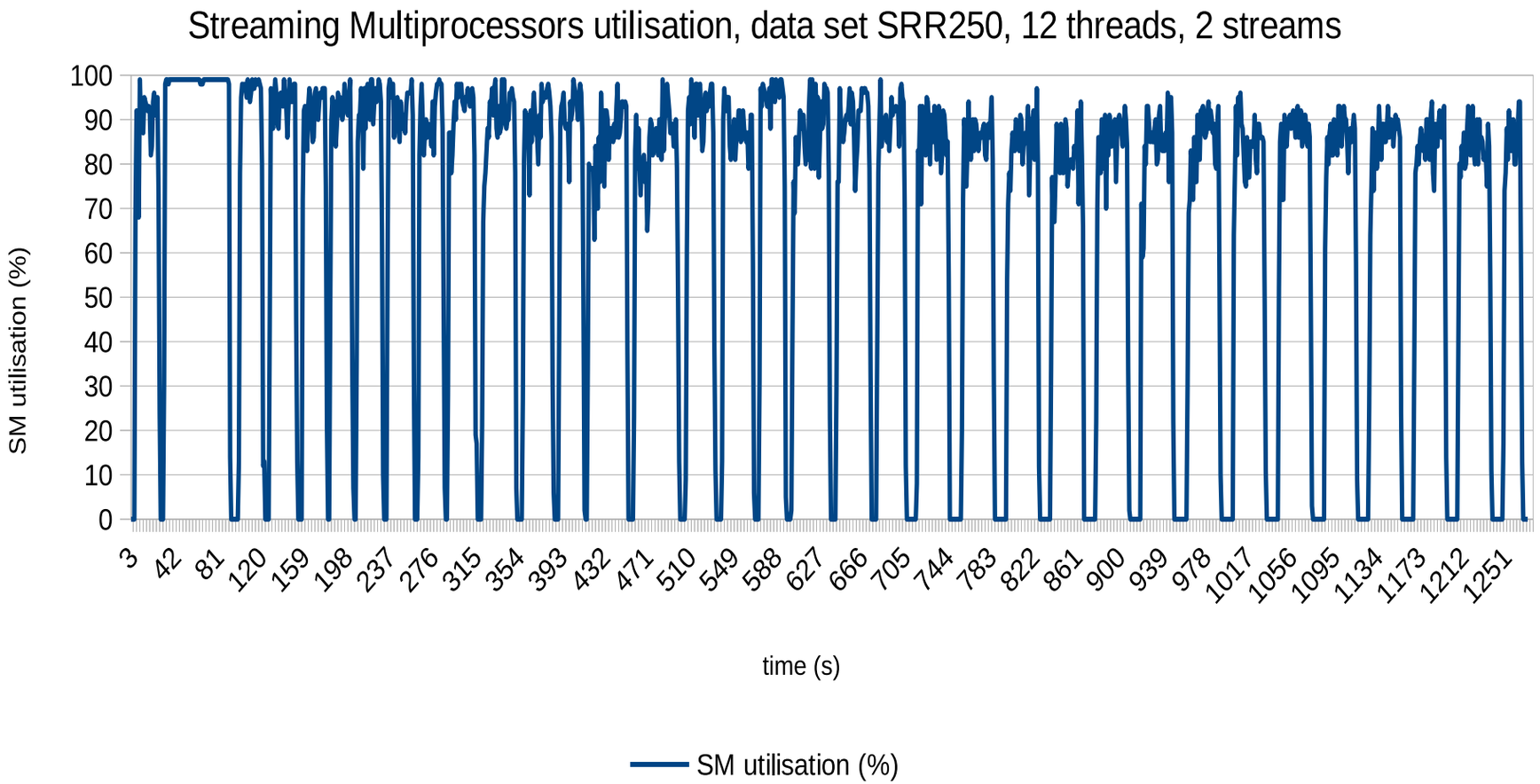
4.4) Results: resources use



4.4) Results: resources use



4.4) Results: resources use



Wrap-up

- Integrated GASAL2 in BWA-MEM
- Effective speed-up **1.28x** (close to max. 1.37x)
- “Seed-only” paradigm to compute **left/right in parallel**
- **Dynamic memory management**, needed for seed-extension
- **98.2%** of main results are exactly matching
- Good use of resources, but there is room for more

Future works

- Optimise kernel
- Enhance GASAL2 compression (3-bit)
- Enhance asymmetric treatment
  - Provide more streams for long batches than shorts
- Accelerate another part of the mapping

Thank you for your attention.