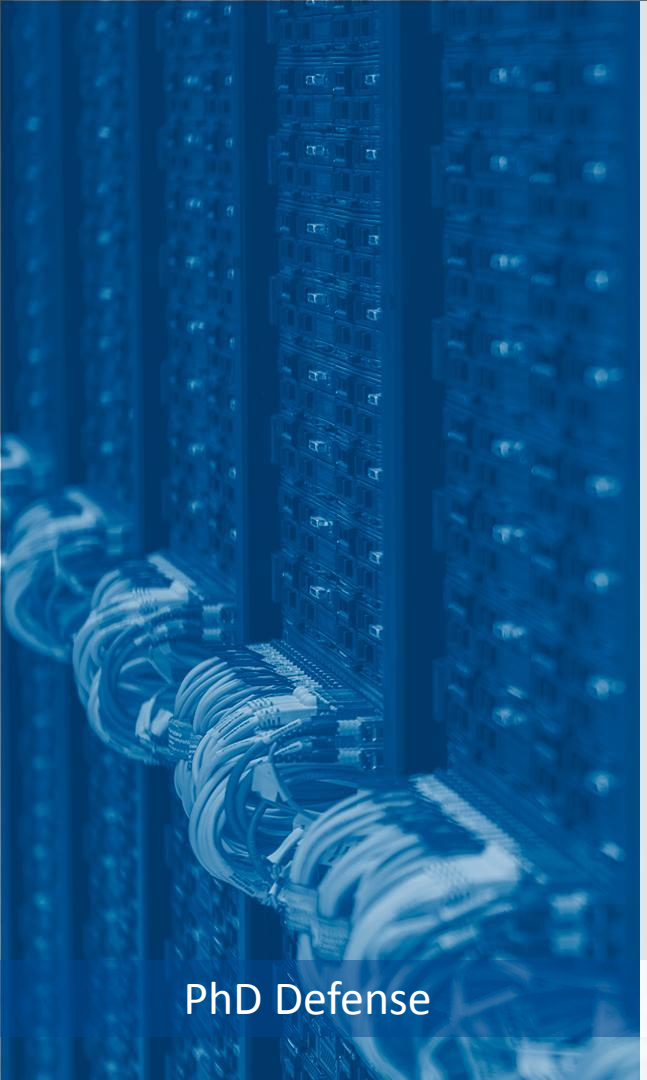


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Universidad Zaragoza

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Improving Performance of Genomics Workloads through Software Optimizations and Hardware Acceleration

Rubén Langarita Benítez

Universitat Politècnica de Catalunya

Barcelona Supercomputing Center

PhD Defense

Adrià Armejach (BSC/UPC) **Jesús Alastruey Benedé** (Unizar)

Why Genomics is Important

- Genomics focuses on the evolution, structure and nature of the genome
- Detect cancer cells
- Differences between monkeys and humans
- Sequencing COVID-19 genome
- The study of the genome is key in personalized medicine to prevent, diagnose, and treat diseases
    - Detection of respiratory diseases
    - Drug usage
    - Cancer treatment

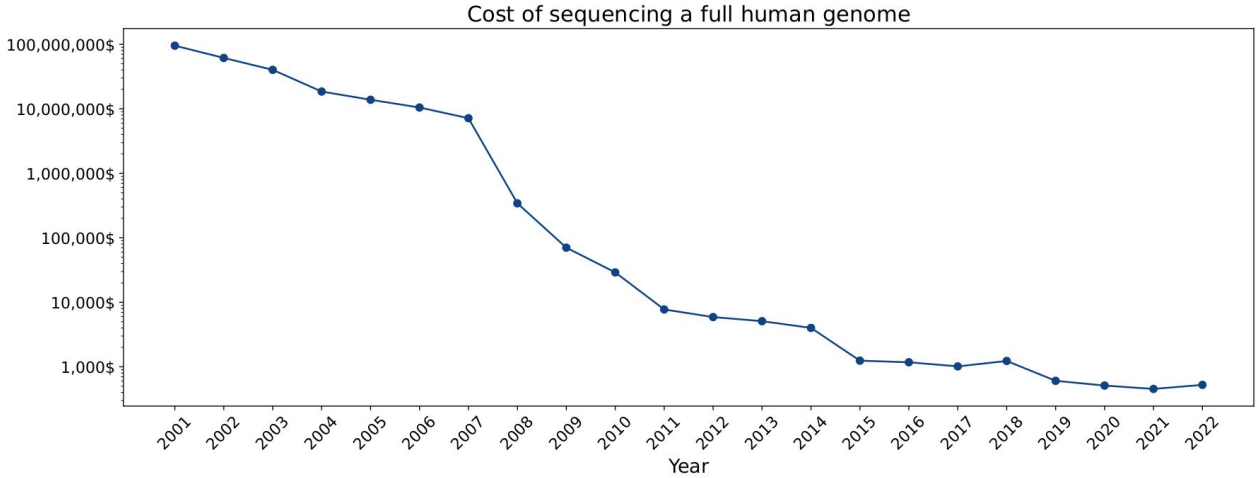
High-level Overview: Sampling

- A biological sample is introduced in the machine: hair or saliva
- A file is obtained with partial reads of the genome, chunks of the entire genome
- In the last decades, these machines have increased their efficiency

```
>SRR622457.3750001 3750001 length=101
CCTCACCTCTATCAAAAATACAAAAATTAGCTGGGCATGGCAGCGGGCACCTGTAACCCCAGCTACTCAGGAGGCTGAGGCAGGAGAATTGCTTGAACCCG
>SRR622457.3750002 3750002 length=101
AACCTCACCTCTATCAAAAATACAAAAATTAGCTGGGCATGGCAGCGGGCACCTGTAACCCCAGCTACTCAGGAGGCTGAGGCAGGAGAATTGCTTGAACC
>SRR622457.3750003 3750003 length=101
GCTACTCAGGAGGCTGAGGCAGGAGAATTGCTTGAACCCCGGAGGCGGAGGCTGCAGTGAGCTGAGCCCAAAGACAGAGGGAGCAGTCAAGTGAGAGAATG
>SRR622457.3750004 3750004 length=101
TGAACCTAGGGATAGAAAGATAGCTGGGTCACAGGAAGAGACTGGAATCAAGAATCCAGGTGCTGCCAGGGCTTCCTGTGTCTGCATCATTATCATTACCC
...
```

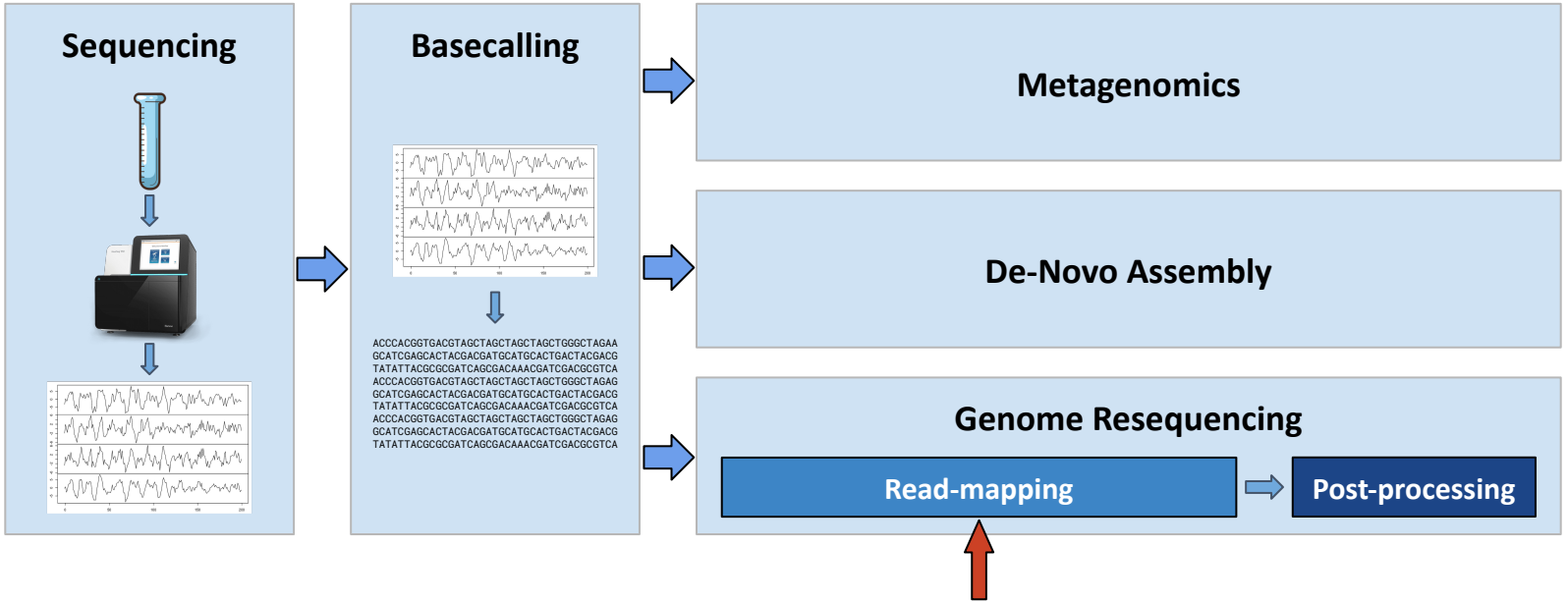
Cost of Sequencing Samples

- Exponential drop in the cost of sequencing
- Exponential increase in data production: More than 1 Tbases/day
- The bottleneck has shifted from the sequencing machines to genomic data processing



Data source: National Human Genome Research Institute (2022)

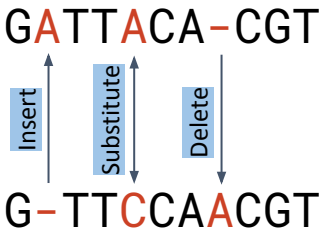
Genomics Pipeline



We focus our efforts on **read-mapping** since it is one of the most compute intensive stages

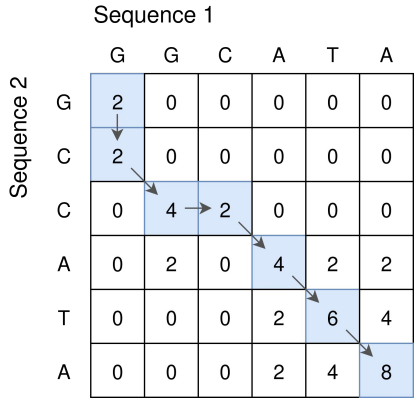
Read-mapping

- The objective is to align the reads against the reference genome
- An **alignment** is a way of arranging two DNA sequences to find their similarities and mutations
- Alignment algorithms transform one sequence into the other using these operations:
  - **Insert** one base-pair
  - **Delete** one base-pair
  - **Substitute** one base-pair with another

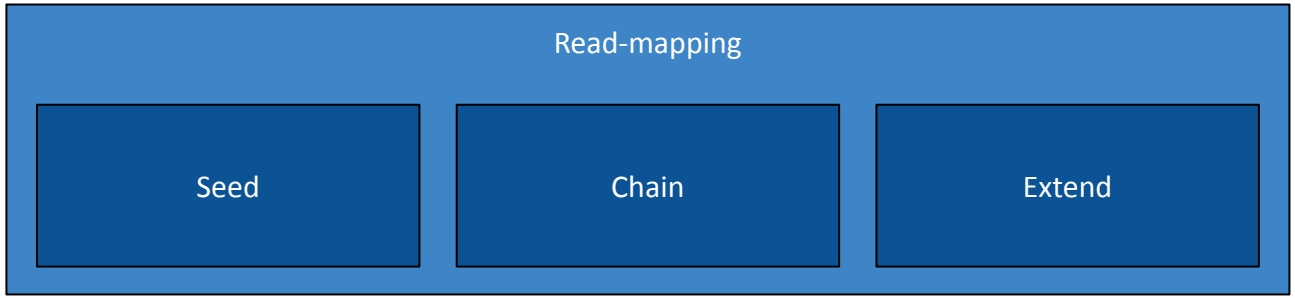


Read-mapping

- A dynamic programming algorithm is used to score the alignments, typically **Smith-Waterman**
- These algorithms have a high computational cost: *O(nm)*
  - Where *n* and *m* are the length of the sequences
- Read-mapping tools need to **avoid aligning against the whole reference genome** to keep computational cost down (human genome: 3 Gbase-pairs)



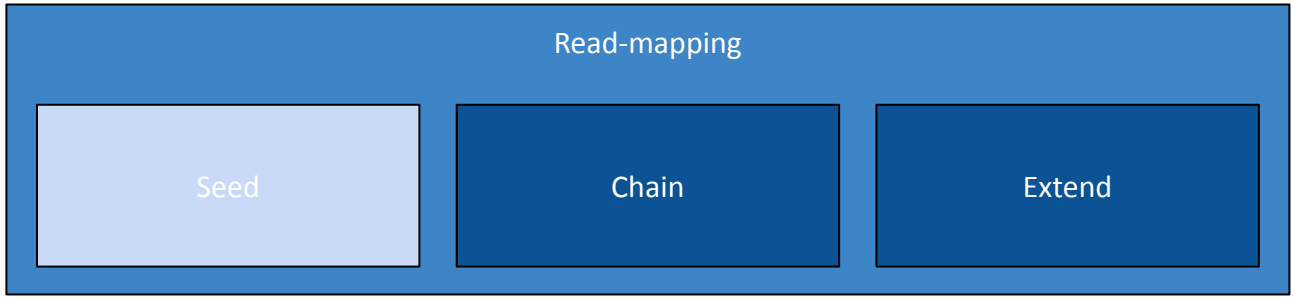
Read-mapping: Seed-and-Extend



To reduce the computation, the **seed-and-extend** strategy is used:

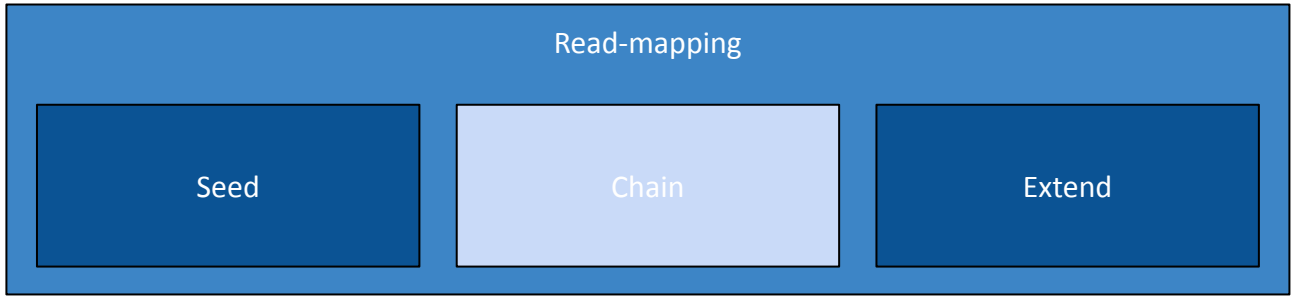
- **Seed**: the objective is to find partial exact matches to reduce the search space
- **Chain** (optional): filter algorithm to reduce the number of seeds
- **Extend**: use dynamic programming algorithms to score the alignments around the seeds

Seed: FM-Index Structure

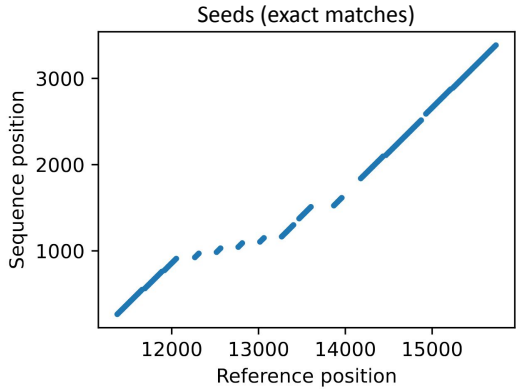


- We search for exact matches between the query and the reference
- It is faster than align the whole reference
- **FM-Index** is a data structure that enables exact search in *O(n)*, where *n* is the length of the searched sequence
- Typically used for short reads (~100 base-pairs)
- The FM-Index is built for the reference genome, e.g., the human genome (~3 gigabase-pairs)

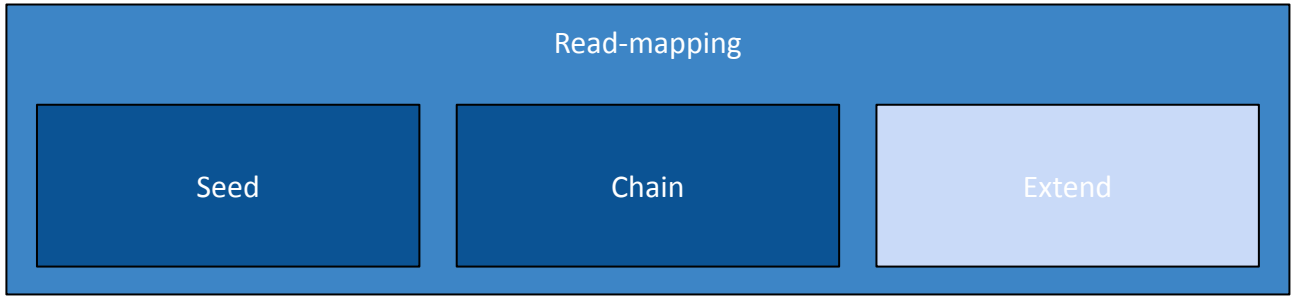
Filtering Seeds: Chain



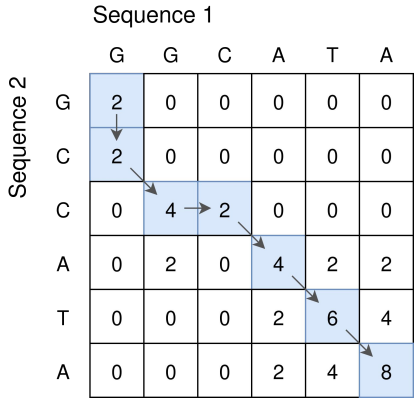
- After seeding, we can filter out meaningless seeds
- **Chain** is a seed filter to reduce computational cost even further
- It is used in the **Minimap2** read-mapping tool
- It combines several seeds (exact matches) to create a more extensive matching region



Extend: Smith-Waterman



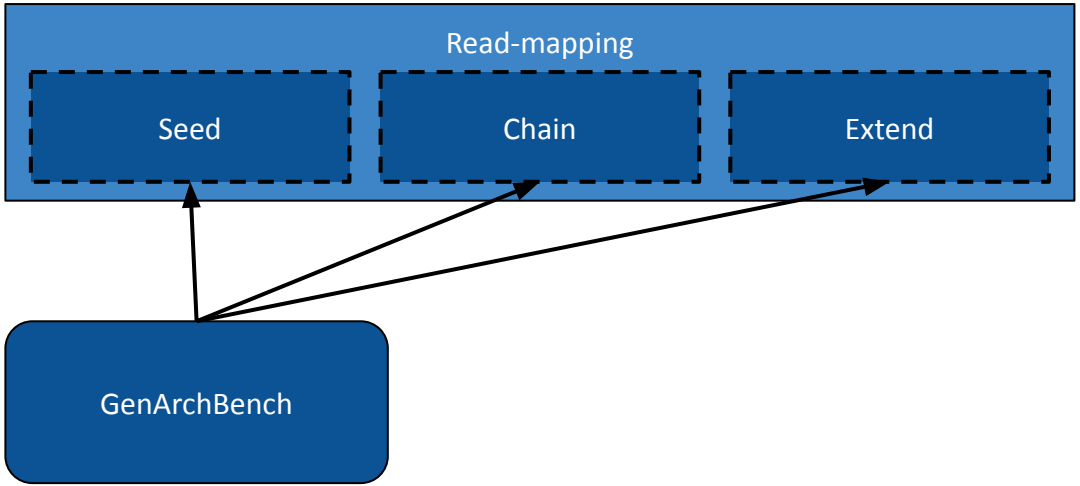
- Now, we have to "extend" the seeds
- The **seeds** are exact matches between the query and the reference genome → **already aligned**
- **Smith-Waterman** algorithm now has to align around the seeds



In this thesis, we focus on:

- Building a **genomic benchmark suite**:
  - Standardize performance measurement
  - Method that can be replicated across modern machines
- **Optimizing genomic algorithms** using a hardware-software codesign
- Proposing a new **hardware accelerator** that targets genomics workloads

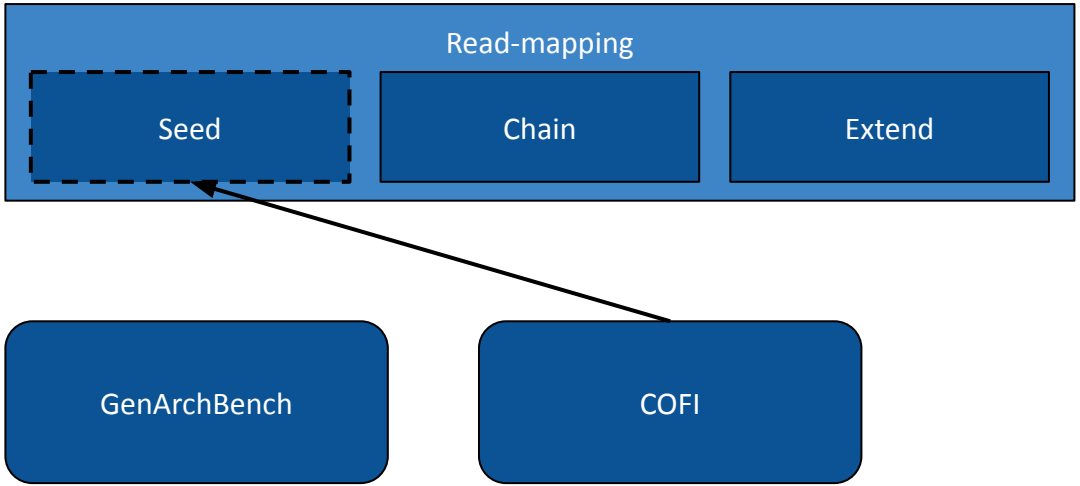
Contribution 1: GenArchBench



GenArchBench: Benchmark suite to enable reproducible benchmarking across systems

We focus on porting and optimizing the **3 kernels present in read-mapping**

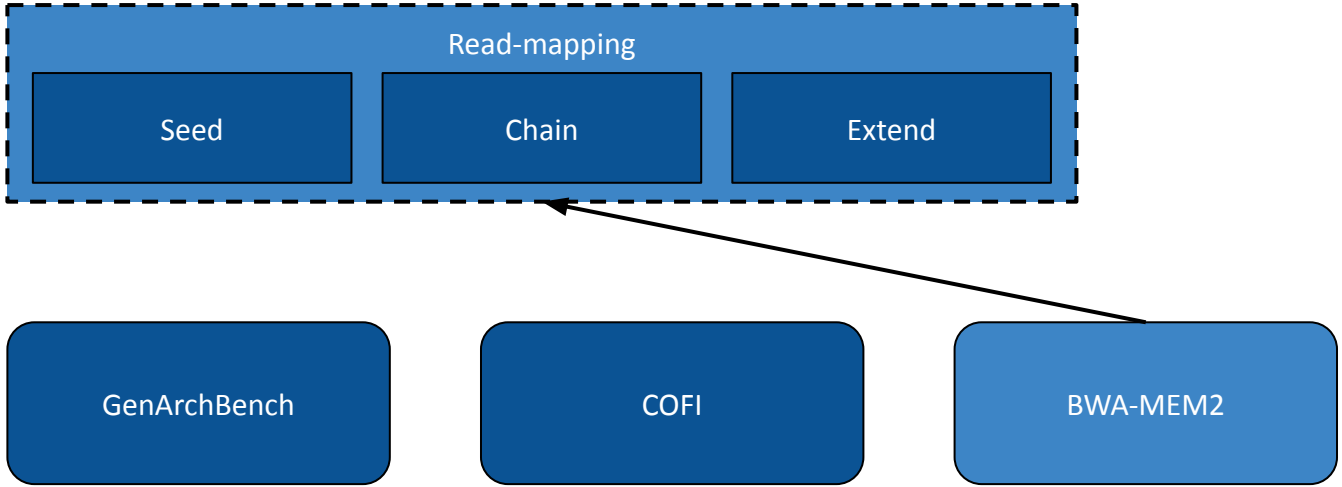
Contribution 2: COFI



COFI: A new compressed FM-Index to accelerate the **seed stage**

We propose a new FM-Index layout that enables searching 15 base-pairs per step

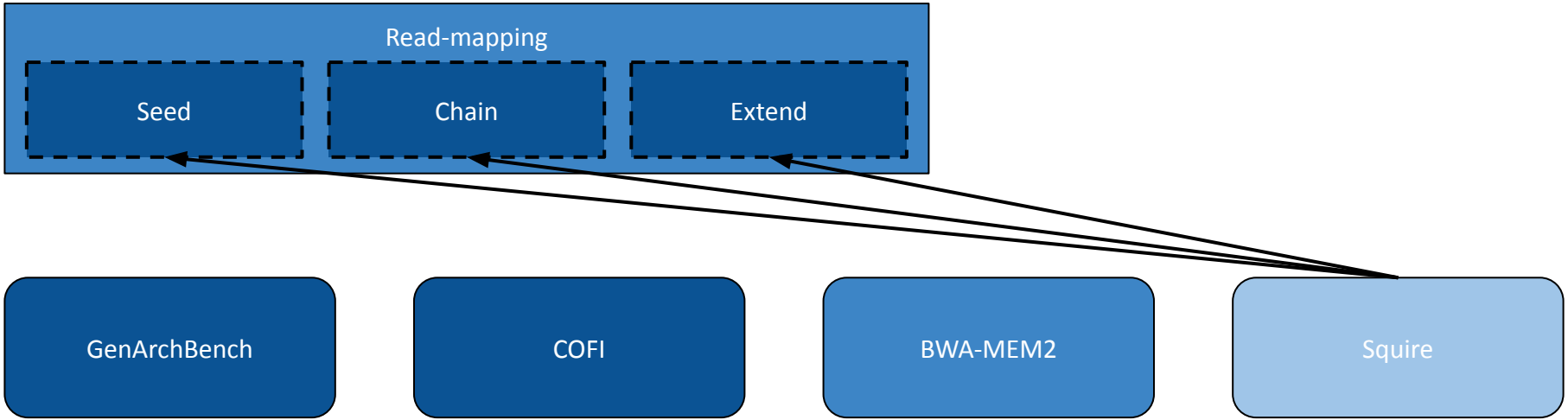
Contribution 3: BWA-MEM2



BWA-MEM2 is a widely-used read-mapping tool

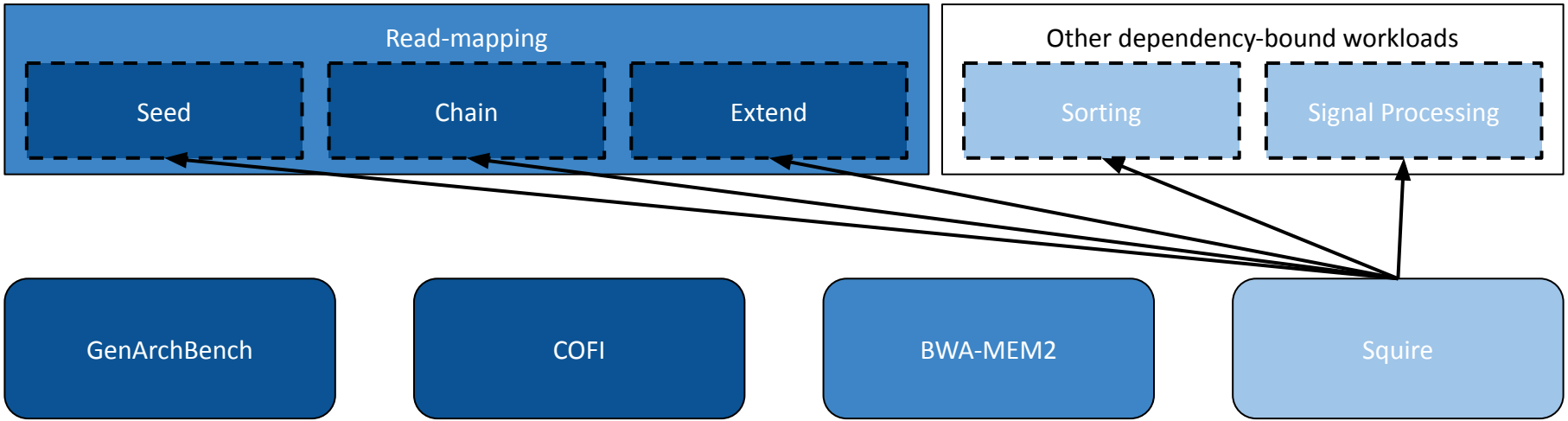
We port and optimize BWA-MEM2 **read-mapping tool** to the ARM architecture

Contribution 4: Squire



Squire is a **hardware accelerator** for genomics kernels

Contribution 4: Squire



We make a general-purpose accelerator for **dependency-bound workloads**

Index

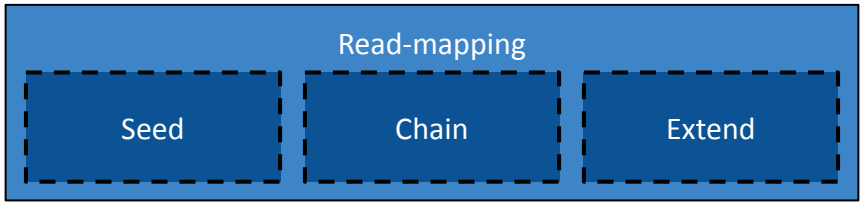
1 - GenArchBench

Genomic Benchmark Suite

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

GenArchBench



- With the rise of ARM architecture in HPC systems, we see the need for a genomics benchmark suite in ARM
- GenArchBench is a benchmark suite that enables comparing different systems and ISAs:
  - Available for x86 and ARM
- 13 computationally-demanding kernels from the most widely-used genomics tools:

ABEA	NN-BASE	**FMI** (Seed)
BPM	NN-VARIANT	**FAST-CHAIN**
CHAIN	PILEUP	**BSW** (Extend)
DBG	POA	
KMER-CNT	WFA	

FM-Index Search (FMI)

- We port (from x86 to ARM) Super Maximal Exact Matches, the algorithm used in the seed stage of BWA-MEM2 that utilizes the FM-Index structure
- We optimize the ported code: 1.35× of speed-up w.r.t. unoptimized code
  - We use the **population count** SVE native instruction (originally implemented using a SWAR approach)
  - **Forcing inline** for one of the most used functions (enables compiler optimization beyond function boundaries)
  - **Interleave sequences** to hide memory latency

SIMD Seed Chaining (FAST-CHAIN)

- Chain: seed filter algorithm used in Minimap2
- FAST-CHAIN is the SIMD version
- We implement FAST-CHAIN using ARM SIMD intrinsics
- Up to 2.2× of speed-up w.r.t. ARM scalar version

```
for (int32_t j = i - 1; j >= st; j-=VL) {
  int32_t real_j = j-VL+1;
  svbool_t valid_elements = svnot_b_z(ptrue,svwhilelt_b32_s32(real_j,st));
  svint32_t rj = svld1_s32(valid_elements,(int32_t*)&anchors_x32[real_j]);
  svint32_t qj = svld1_s32(valid_elements,(int32_t*)&anchors_y32[real_j]);

  svint32_t dr = svsub_s32_x(valid_elements,ri,rj);
  svint32_t dq = svsub_s32_x(valid_elements,qi,qj);
  svint32_t dd = svabd_s32_x(valid_elements,dr,dq);

  svbool_t skip_anchor = svcmpeq_n_s32(valid_elements,dr,0);
  valid_elements = svbic_b_z(valid_elements,valid_elements,skip_anchor);
  skip_anchor = svcmple_n_s32(valid_elements,dq,0);
  valid_elements = svbic_b_z(valid_elements,valid_elements,skip_anchor);
  skip_anchor = svcmpgt_n_s32(valid_elements,dq,max_dist_y);
  valid_elements = svbic_b_z(valid_elements,valid_elements,skip_anchor);
  skip_anchor = svcmpgt_n_s32(valid_elements,dq,max_dist_x);
  valid_elements = svbic_b_z(valid_elements,valid_elements,skip_anchor);
  skip_anchor = svcmpgt_n_s32(valid_elements,dd,bw);
  valid_elements = svbic_b_z(valid_elements,valid_elements,skip_anchor);

  if (!svptest_any(ptrue,valid_elements)) continue;

  svint32_t dr_dq_min = svmin_s32_x(valid_elements,dr,dq);
  svint32_t oc = svmin_n_s32_x(valid_elements,dr_dq_min,q_spani);

  svbool_t valid_log = svcmpne_n_s32(valid_elements,dd,0);
  svint32_t log_dd = svreinterpret_s32(svclz_s32_x(valid_elements,dd));
  log_dd = svsubr_n_s32_z(valid_log,log_dd,31);

  svfloat32_t gap_cost_f = svcvt_f32_s32_x(valid_elements,dd);
  gap_cost_f = svmul_n_f32_x(valid_elements,gap_cost_f,avg_qspan001);
  svint32_t gap_cost = svcvt_s32_f32_x(valid_elements,gap_cost_f);
  log_dd = svreinterpret_s32(svlsr_n_u32_x(valid_elements,svreinterpret_u32(log_dd),1));
  gap_cost = svadd_s32_x(valid_elements,gap_cost,log_dd);

  svint32_t score = svld1_s32(valid_elements,&scores[real_j]);
  score = svadd_s32_x(valid_elements,score,oc);
  score = svsub_s32_x(valid_elements,score,gap_cost);

  int32_t max_local = svmaxv_s32(valid_elements,score);
}
```

Banded Smith-Waterman (BSW)

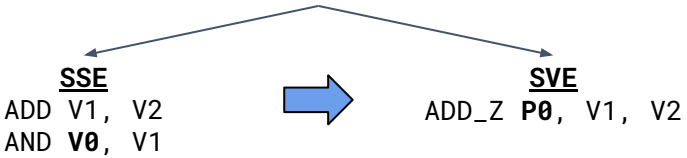
- We port Banded Smith-Waterman, the alignment algorithm used in the extend stage of BWA-MEM2, from SSE (x86 SIMD) to SVE (ARM SIMD)

SSE intrinsic	SVE translation	Functionality
_mm_malloc	aligned_alloc	Allocate memory
_mm_free	free	Free memory
__rdtsc	cntvct_el0	Count cycles
_mm_prefetch	__builtin_prefetch	Software prefetch
_mm_setzero_si128	svdup_s64(0)	Set the register to zero
_mm_set1_epi{8,16}	svdup_s{8,16}	Set the register to a given value
_mm_blend_epi{8,16}	svsel	Select from two registers with a predicate
_mm_add_epi{8,16}	svadd_x	Addition
_mm_adds_epu{8,16}	svqadd	Addition with saturation
_mm_sub_epi{8,16}	svsub_x	Subtraction
_mm_subs_ep{i,u}{8,16}	svqsub	Subtraction with saturation
_mm_max_ep{i,u}{8,16}	svmax_x	Maximum
_mm_min_epu{i,u}{8,16}	svmin_x	Minimum
_mm_and_si128 (arithmetic)	svand_z	Bit-wise AND
_mm_and_si128 (predicate)	svand_x	Logical AND
_mm_or_si128 (arithmetic)	svorr_z	Bit-wise OR
_mm_or_si128 (predicate)	svorr_x	Logical OR
_mm_xor_si128 (arithmetic)	sveor_x	Bit-wise XOR
_mm_andnot_si128 (arithmetic)	svbic_x	Bit-wise ANDNOT
_mm_andnot_si128 (predicate)	svbic_z	Logical ANDNOT
_mm_cmpeq_epi{8,16}	svcmpeq	Equal comparison
_mm_cmpgt_epi{8,16}	svcmpgt	Greater than comparison
_mm_cmpge_epi16	svcmpge	Greater or equal comparison
_mm_load_si128	svld1	Memory load
_mm_store_si128	svst1	Memory store

- We convert SSE masking to SVE predication

```
if (y < 0) x = y + z
else x = 0
```

```
V0/P0 = CMPLT V1, #0
```

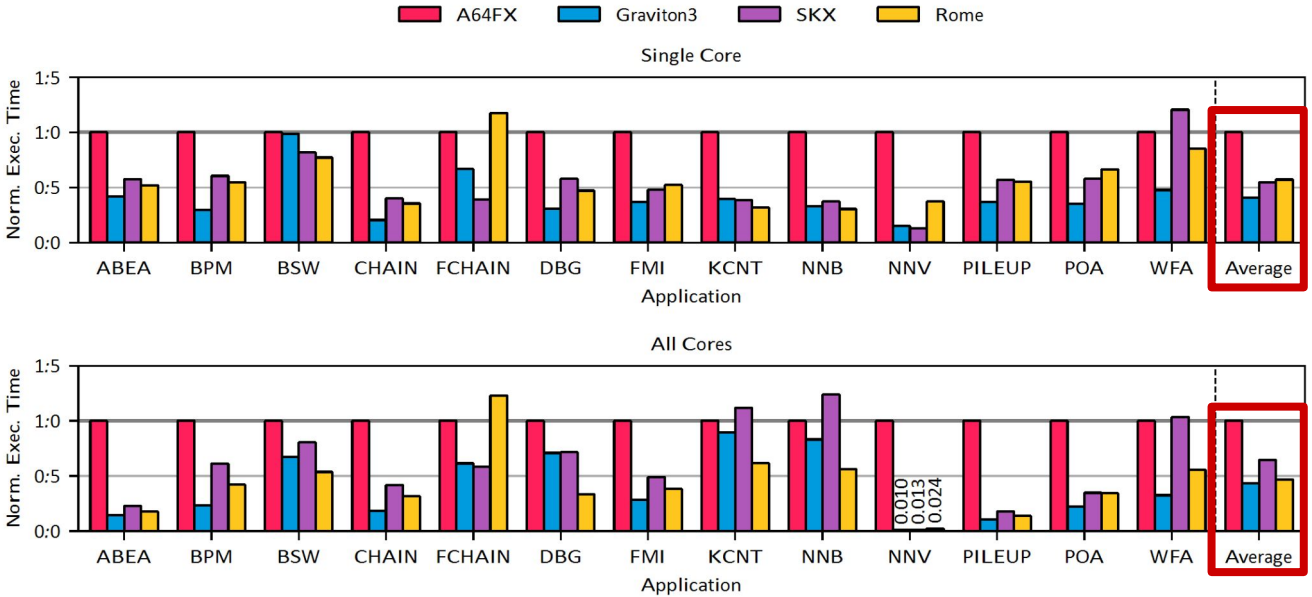


- Up to 3.4× of speed-up w.r.t. ARM scalar version

Methodology: Hardware Platforms

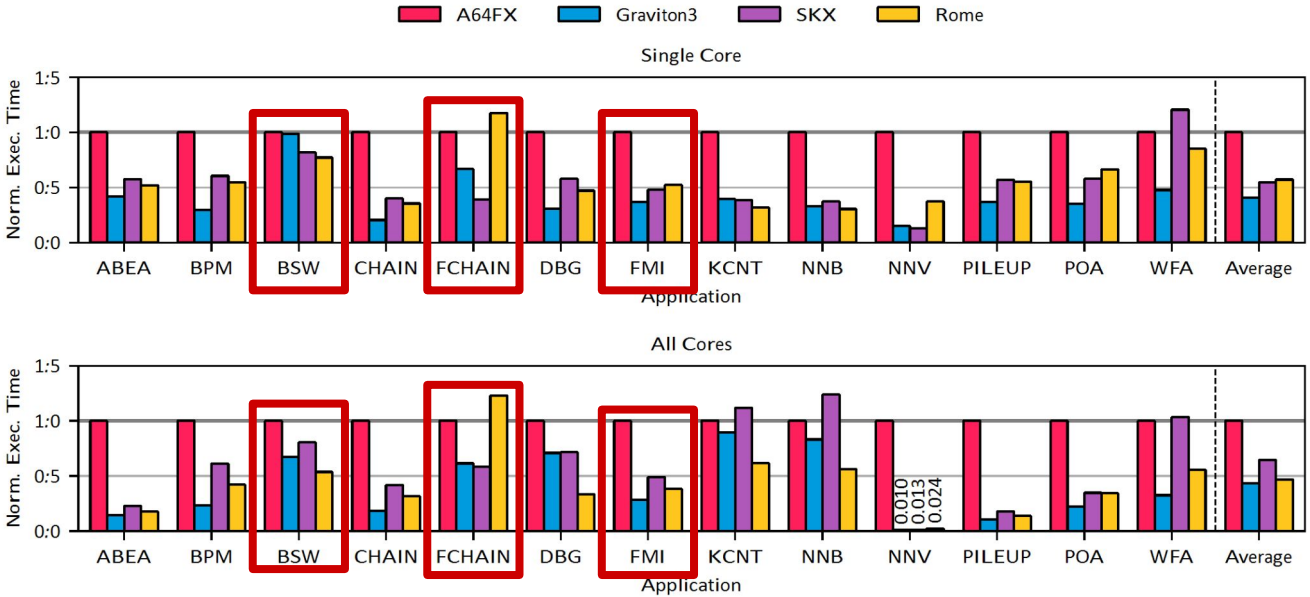
	A64FX (ARM)	Graviton3 (ARM)	Intel SKX (x86)	AMD Rome (x86)
Cores	4 × 12	64	2 × 24	64
Frequency	2.2 GHz	2.6 GHz	1 - 2.1 GHz	1.5 - 2.25 GHz
Memory capacity	4 × 8 GB	128 GB	2 × 48 GB	1024 GB
Memory technology	on-package HBM2	off-package DDR5 4800 MHz	off-package DDR4 2667 MHz	off-package DDR4 3200 MHz
Peak bandwidth	4 × 256 GB/s	300 GB/s	2 × 120 GB/s	204.8 GB/s
Vector extension	NEON/SVE 512 bits	NEON/SVE 256 bits	SSE/AVX2/AVX512	SSE/AVX2

Evaluation



- **Graviton 3 (ARM)** beats the other systems → More cores (64) and more out-of-order resources
- **SKX (x86)** beats Rome in single thread → Larger vector width (512 bits vs 256 bits)
- **Rome (x86)** beats SKX when using all threads → More cores (64 vs 48)
- **A64FX (ARM)** presents the worst performance → Higher memory latency and fewer out-of-order resources

Evaluation



- **A64FX**:
  - Performs relatively better in **BSW** and **FCHAIN** (512-bit vector width and HBM2)
  - Worse performance in **FMI** (higher memory latency)
- **Graviton 3**:
  - **BSW** and **FCHAIN** take advantage from SIMD, Graviton 3 is beaten by other systems (256-bit vector width)
  - Outperforms other systems in **FMI** (lower memory latency)

GenArchBench Conclusions

- We port and evaluate 13 genomics kernels to ARM
- I contribute to GenArchBench by porting and optimizing 3 kernels from read-mapping tools
- We evaluate the 13 kernels in 2 ARM machines and 2 x86 machines

https://github.com/LorienLV/genarchbench/releases/tag/1.0.0

Lorién López-Villellas, **Rubén Langarita**, Asaf Badouh, Víctor Soria-Pardos, Quim Aguado-Puig, Guillem López-Paradís, Max Doblas, Javier Setoain, Chulho Kim, Makoto Ono, Adrià Armejach, Santiago Marco-Sola, Jesús Alastruey-Benedé, Pablo Ibáñez, and Miquel Moretó (2024). *GenArchBench: A genomics benchmark suite for arm HPC processors*. Future Generation Computer Systems, 157, 313-329.

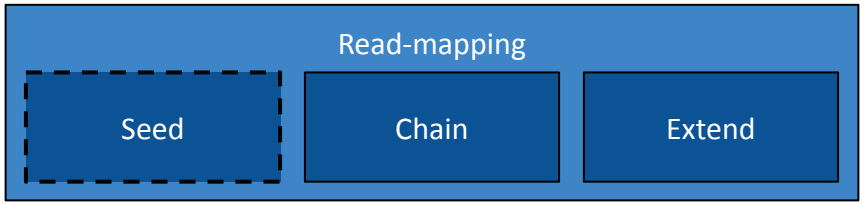
2 - COFI

Optimized Data Structure for Exact Search

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

FM-Index



- We focus on **seed** → finding exact matches
- We propose a new data layout for **FM-Index**
- Algorithmic modifications to improve search throughput of exact matches

How to Build FM-Index

M

0	A	G	A	T	G	C	C	A	G	G	C	C	A	T	$
1	G	A	T	G	C	C	A	G	G	C	C	A	T	$	A
2	A	T	G	C	C	A	G	G	C	C	A	T	$	A	G
3	T	G	C	C	A	G	G	C	C	A	T	$	A	G	A
4	G	C	C	A	G	G	C	C	A	T	$	A	G	A	T
5	C	C	A	G	G	C	C	A	T	$	A	G	A	T	G
6	C	A	G	G	C	C	A	T	$	A	G	A	T	G	C
7	A	G	G	C	C	A	T	$	A	G	A	T	G	C	C
8	G	G	C	C	A	T	$	A	G	A	T	G	C	C	A
9	G	C	C	A	T	$	A	G	A	T	G	C	C	A	G
10	C	C	A	T	$	A	G	A	T	G	C	C	A	G	G
11	C	A	T	$	A	G	A	T	G	C	C	A	G	G	C
12	A	T	$	A	G	A	T	G	C	C	A	G	G	C	C
13	T	$	A	G	A	T	G	C	C	A	G	G	C	C	A
14	$	A	G	A	T	G	C	C	A	G	G	C	C	A	T

M | **BWT** | **SA**

$	A	G	A	T	G	C	C	A	G	G	C	C	A	T	14
A	G	A	T	G	C	C	A	G	G	C	C	A	T	$	0
A	G	G	C	C	A	T	$	A	G	A	T	G	C	C	7
A	T	$	A	G	A	T	G	C	C	A	G	G	C	C	12
A	T	G	C	C	A	G	G	C	C	A	T	$	A	G	2
C	A	G	G	C	C	A	T	$	A	G	A	T	G	C	6
C	A	T	$	A	G	A	T	G	C	C	A	G	G	C	11
C	C	A	G	G	C	C	A	T	$	A	G	A	T	G	5
C	C	A	T	$	A	G	A	T	G	C	C	A	G	G	10
G	A	T	G	C	C	A	G	G	C	C	A	T	$	A	1
G	C	C	A	G	G	C	C	A	T	$	A	G	A	T	4
G	C	C	A	T	$	A	G	A	T	G	C	C	A	G	9
G	G	C	C	A	T	$	A	G	A	T	G	C	C	A	8
T	$	A	G	A	T	G	C	C	A	G	G	C	C	A	13
T	G	C	C	A	G	G	C	C	A	T	$	A	G	A	3

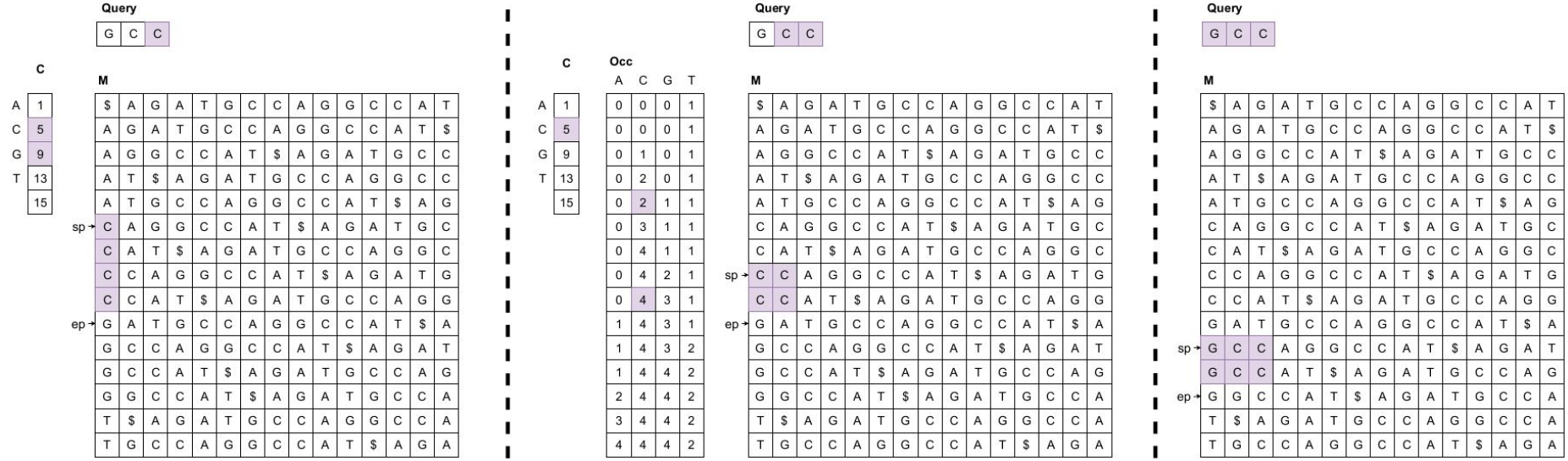
C

A	1
C	5
G	9
T	13
	15

Occ

A	C	G	T
0	0	0	1
0	0	0	1
0	1	0	1
0	2	0	1
0	2	1	1
0	3	1	1
0	4	1	1
0	4	2	1
0	4	3	1
1	4	3	1
1	4	3	2
1	4	4	2
2	4	4	2
3	4	4	2
4	4	4	2

Search Using FM-Index



$$sp = C['C'] + Occ[sp, 'C']$$

- Occ structure takes several GBs for the human genome
- Accesses are unpredictable
- Iteration *i* depends on iteration *i-1*

K-Step FM-Index

BWT	Occ k=1			
	A	C	G	T
T	0	0	0	1
$	0	0	0	1
C	0	1	0	1
C	0	2	0	1
G	0	2	1	1
C	0	3	1	1
C	0	4	1	1
G	0	4	2	1
G	0	4	3	1
A	1	4	3	1
T	1	4	3	2
G	1	4	4	2
A	2	4	4	2
A	3	4	4	2
A	4	4	4	2

BWT		Occ k=2								
		AA	AC	AG	AT	CA	.....	TG	TT
A	T	0	0	0	1	0	...	0	0
T	$	0	0	0	1	0	...	0	0
C	C	0	0	0	1	0	...	0	0
C	C	0	0	0	1	0	...	0	0
A	G	0	0	1	1	0	...	0	0
G	C	0	0	1	1	0	...	0	0
G	C	0	0	1	1	0	...	0	0
T	G	0	0	1	1	0	...	1	0
G	G	0	0	1	1	0	...	1	0
$	A	0	0	1	1	0	...	1	0
A	T	0	0	1	2	0	...	1	0
A	G	0	0	2	2	0	...	1	0
C	A	0	0	2	2	1	...	1	0
C	A	0	0	2	2	2	...	1	0
G	A	0	0	2	2	2	...	1	0

- Increase alphabet size, and the bases searched per iteration
- 2-step FM-Index:
  - Memory footprint is **multiplied by 4**
  - Potential speed-up of **2×**

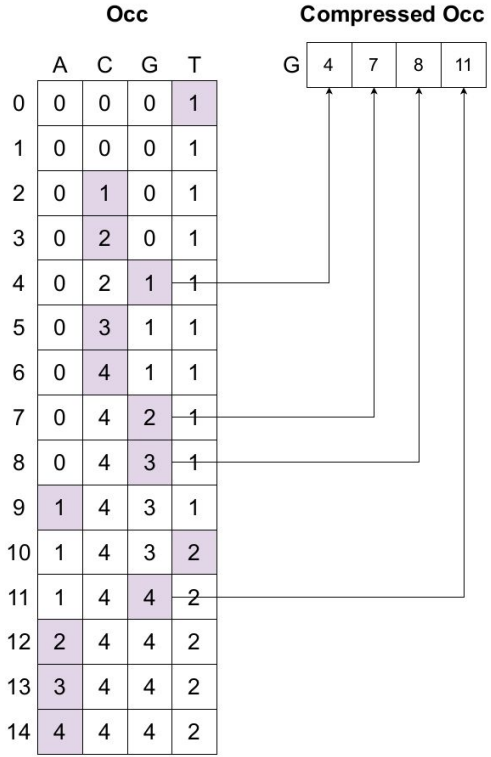
FM-Index Memory Footprint Analysis

- K-step has a potential speed-up of *k*
- FM-Index size increases exponentially with *k*
- However, in each row of Occ, only one counter is increased by one, i.e., 1 out of 4^k
- The rest of the 4^k counters have the same value than the previous row
- What if we **only store the indexes where the counter changes**?

Occ

A	C	G	T
0	0	0	1
0	0	0	1
0	1	0	1
0	2	0	1
0	2	1	1
0	3	1	1
0	4	1	1
0	4	2	1
0	4	3	1
1	4	3	1
1	4	3	2
1	4	4	2
2	4	4	2
3	4	4	2
4	4	4	2

COFI: Compressed FM-Index

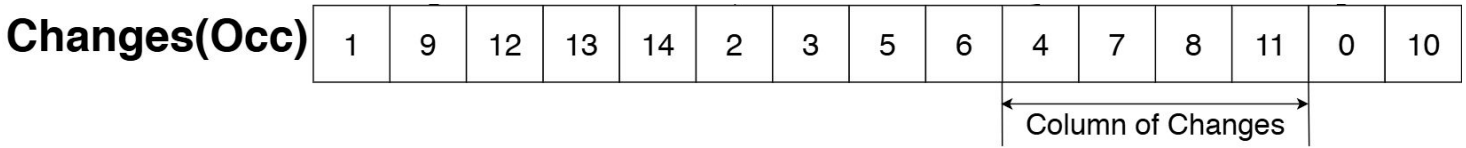


We store the indexes where the counter changes

$$sp = C[\text{`G'}] + Occ[sp, \text{`G'}]$$

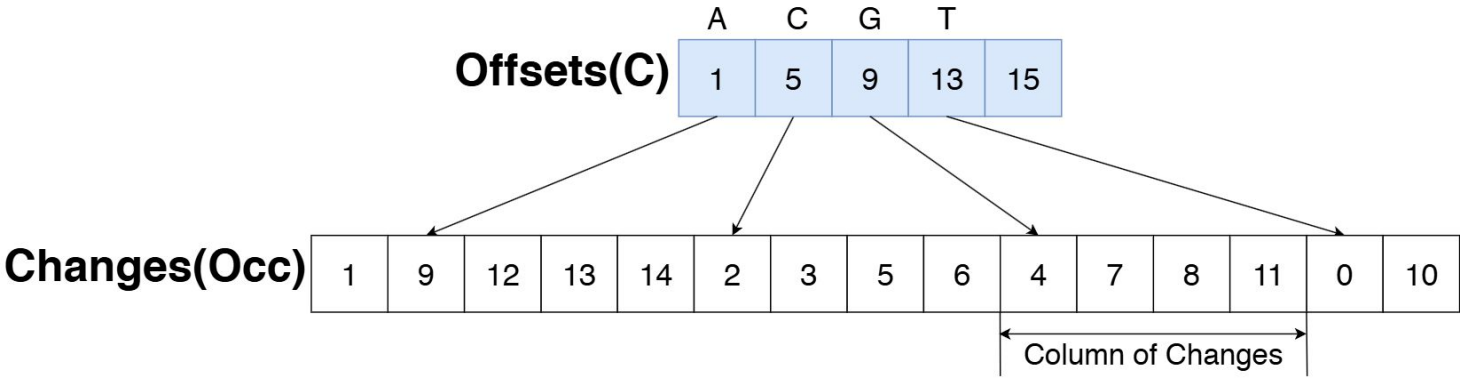
COFI: Compressed FM-Index

- COFI is a **CO**mpressed **F**m-**I**ndex that enables large k-steps



COFI: Compressed FM-Index

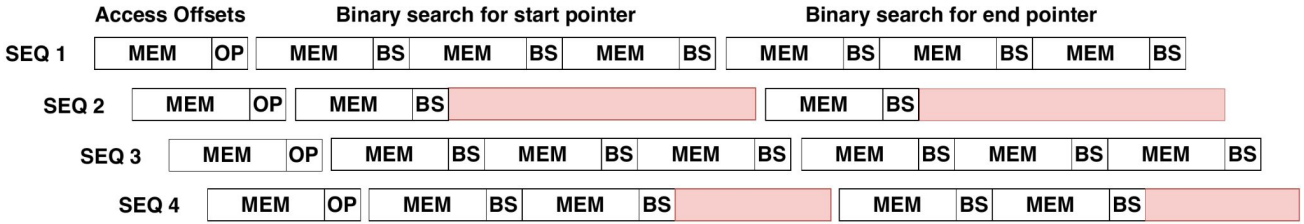
- COFI is a **CO**mpressed **F**m-**I**ndex that enables large k-steps
- Offsets array (previously C) indicates the start and end of columns



- For the human genome:
  - 15-step COFI: 16 GB
  - 15-step SoA FM-Index: order of exabytes (10^{18})

Optimizations on Top of COFI

1. Sequence interleaving



2. Software prefetching

- Prefetch operations are issued for the next symbol to retrieve from memory the necessary *Offsets* elements
- Additionally, the first pivot point for *sp*'s binary search is prefetched once the column to be accessed is known

3. Conditional moves

- Branches are difficult to predict in binary search
- We replace the branches with conditional moves

BRANCH

```
if (pivot < elem)
    left = mid + 1;
```

CMOV

```
left = (pivot < elem) ? mid+1 : left;
```

Methodology: Hardware Platforms

	Intel Xeon Phi 7230 (KNL)	Intel Xeon Platinum 8160 (SKX)
Cores × Threads	64 × 4	24 × 2
Issue width	2	4
Frequency	1.3 GHz	2.1 GHz
MCDRAM	16 GB & 480 GB/s	-
DDR	192 GB & 90 GB/s	96 GB & 120 GB/s

Methodology: Inputs

We use *GRCh38* human genome as the reference: 3.05 gigabase-pairs

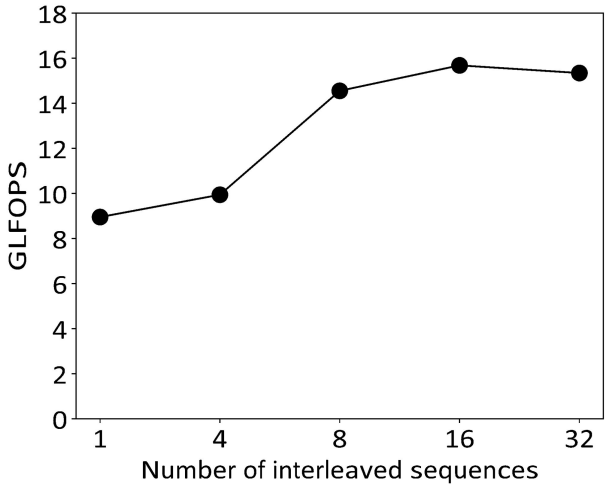
Input	Number of sequences	Length	Occu. GRCh38
sanger	20M	200	46.71M
ocily7-s	35.21M	101	26.41M
ocily7-1	70.38M	101	54.45M
ocily7-2	70.89M	101	39.21M
a375-1	115.32M	101	79.81M
a375-2	115.27M	101	106.96M
mason1	10M	150	384,139
mason2	10M	150	385,207
mason3	10M	150	3,431
mason4	10M	150	3,402
mason5	10M	150	3,407

Evaluation Methodology

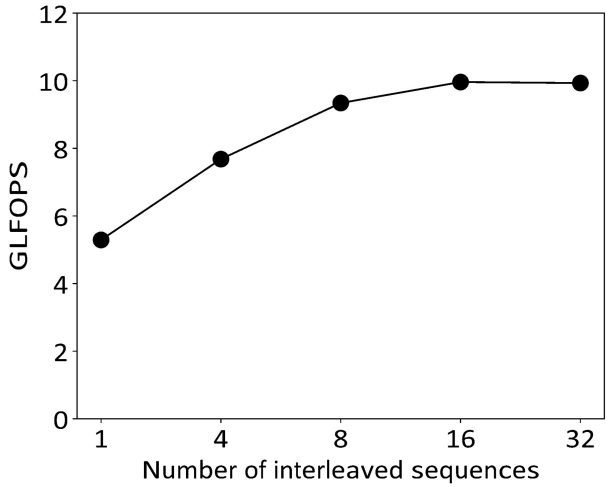
Evaluation in two phases:

- Tuning performance **optimization** parameters:
  - Interleave sequences
  - Software prefetching
  - Conditional moves
- Comparison with **SoA FM-Index**: bit-vector Sampled FM-Index (bvSFM)

Optimization Evaluation: Sequence Interleaving



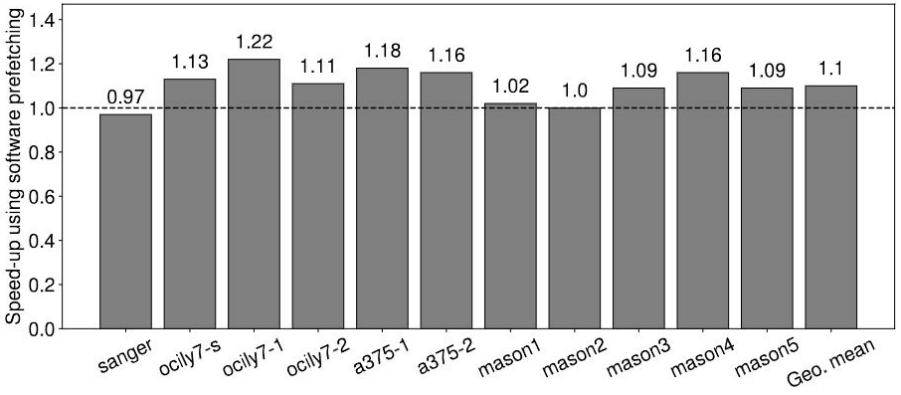
KNL



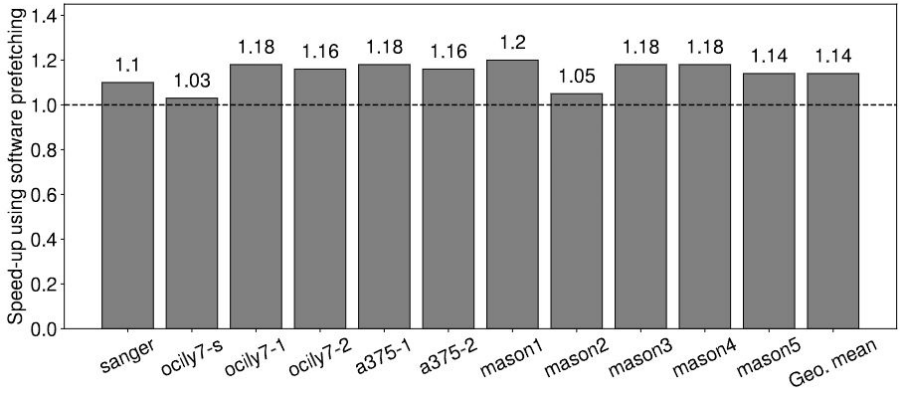
SKX

We achieve the best performance with 16 interleaved sequences

Optimization Evaluation: Software Prefetching



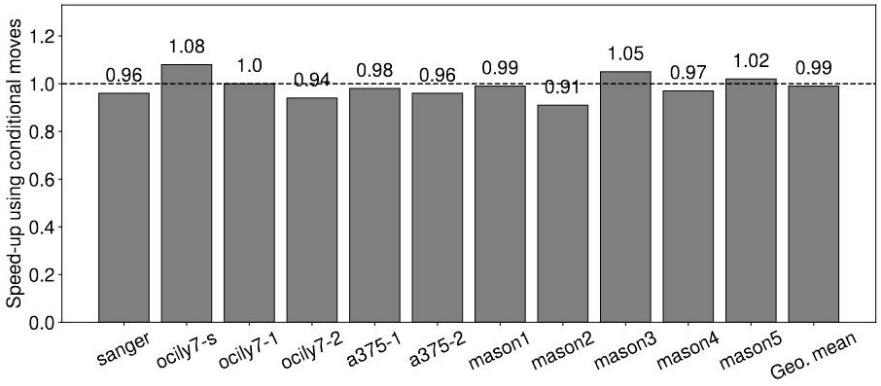
KNL



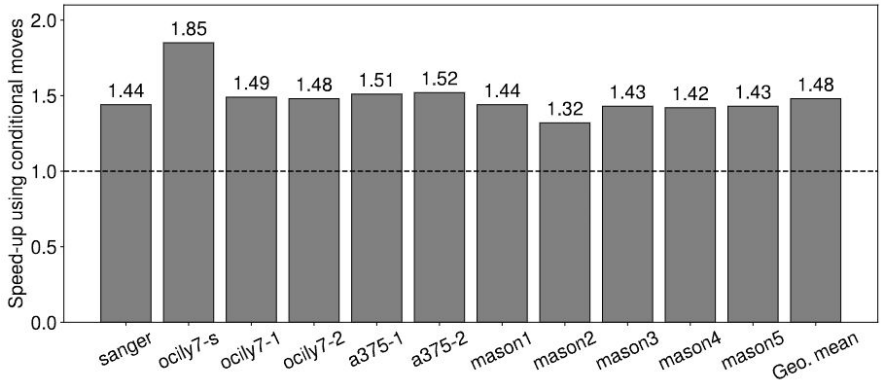
SKX

10% and 14% of speed-up on average

Optimization Evaluation: Conditional Moves



KNL

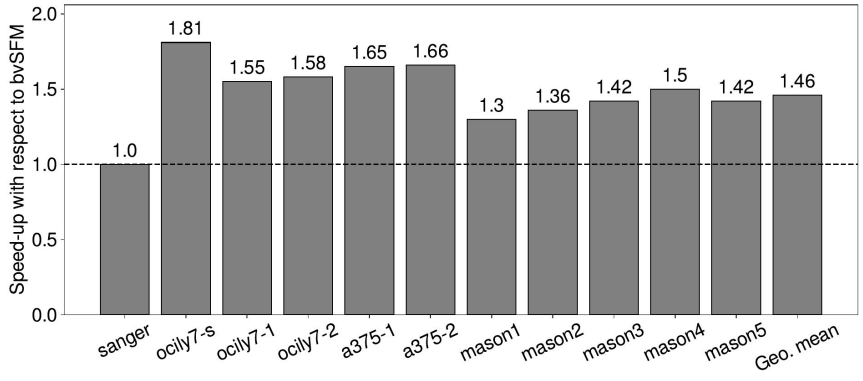


SKX

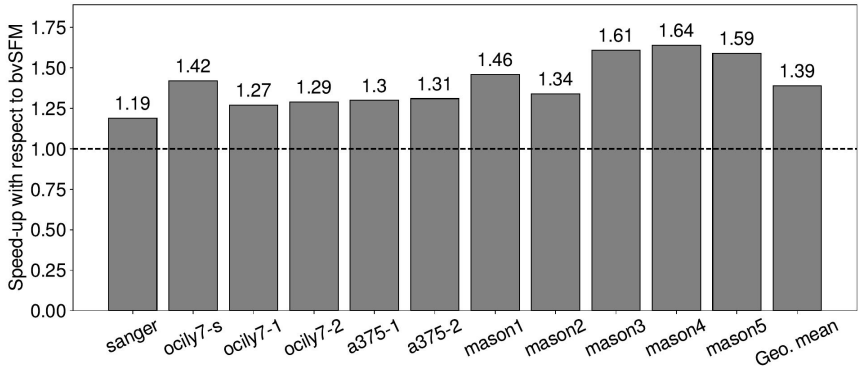
Conditional moves harm KNL → We do not use them in KNL

SKX benefits from conditional moves since it has a deeper pipeline

COFI vs SoA



KNL



SKX

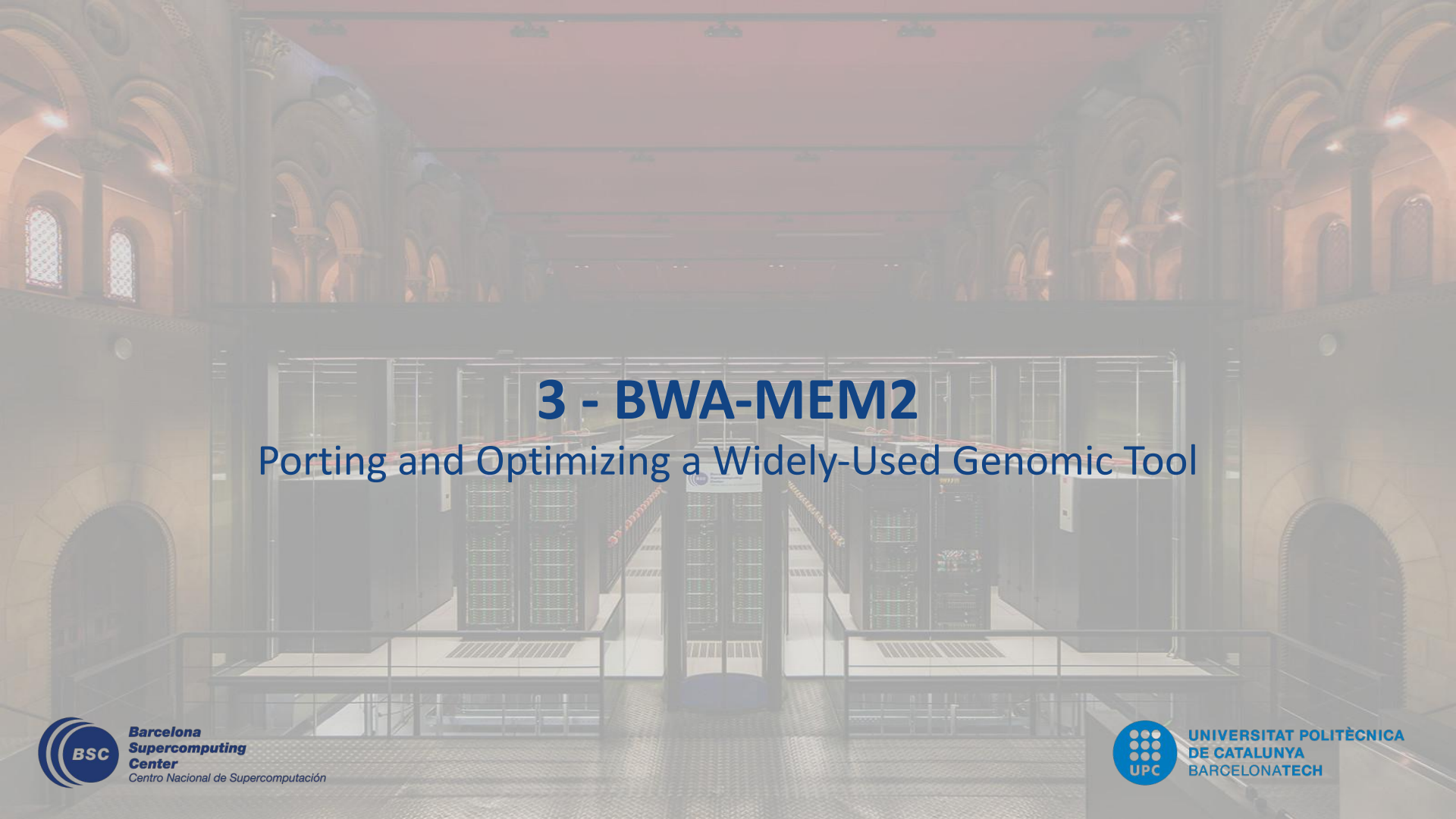
46% and 39% of speed-up on average w.r.t. state-of-the-art (bvSFM)

COFI Conclusions

- COFI is a **CO**mpressed **F**m-**I**ndex used in exact search
- It enables 15-step FM-Index with reasonable memory footprint
- Speed-up of up to 2.14× w.r.t. state-of-the-art

https://gitlab.bsc.es/rlangari/cofi

Rubén Langarita, Adrià Armejach, Javier Setoain, Pablo Ibáñez-Marín, Jesús Alastruey-Benedé, and Miquel Moretó (2020). *Compressed sparse FM-index: Fast sequence alignment using large K-steps*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 19(1), 355-368.



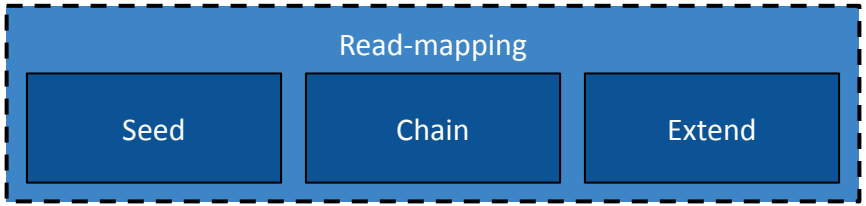
3 - BWA-MEM2

Porting and Optimizing a Widely-Used Genomic Tool

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

BWA-MEM2



- Widely-used read-mapper for short reads (~100 base-pairs)
- Stages:
  - (Seed) Super Maximal Exact Matches (**SMEM**): Uses FM-Index to find seeds
  - (Seed) Suffix Array Look-up (**SAL**): Translates FM-Index positions to reference positions
  - (Extend) Banded Smith-Waterman (**BSW**): Aligns sequences around the seeds

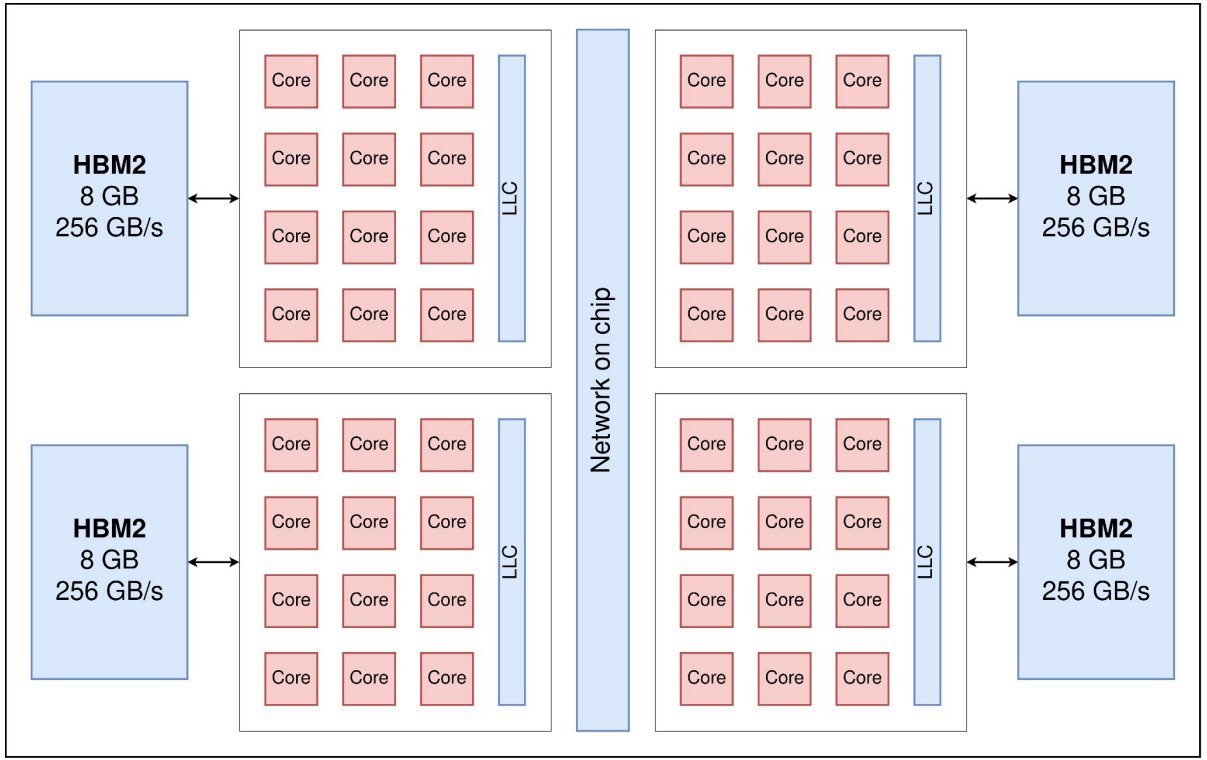
Porting BWA-MEM2 to ARM

- SIMD extensions:
  - **SSE**: x86 vectorial extension, vector is 128-bit wide
  - **SVE**: ARM vectorial extension, vector length depends on hardware implementation (from 128 to 2048)
- BSW is implemented using SSE intrinsics
- We port the SSE version to SVE

SSE intrinsic	SVE translation	Functionality
_mm_malloc	aligned_alloc	Allocate memory
_mm_free	free	Free memory
__rdtsc	cntvct_el0	Count cycles
_mm_prefetch	__builtin_prefetch	Software prefetch
_mm_setzero_si128	svdup_s64(0)	Set the register to zero
_mm_set1_epi{8,16}	svdup_s{8,16}	Set the register to a given value
_mm_blend_epi{8,16}	svsel	Select from two registers with a predicate
_mm_add_epi{8,16}	svadd_x	Addition
_mm_adds_epu{8,16}	svqadd	Addition with saturation
_mm_sub_epi{8,16}	svsub_x	Subtraction
_mm_subs_ep{i,u}{8,16}	svqsub	Subtraction with saturation
_mm_max_ep{i,u}{8,16}	svmax_x	Maximum
_mm_min_epu{i,u}{8,16}	svmin_x	Minimum
_mm_and_si128 (arithmetic)	svand_z	Bit-wise AND
_mm_and_si128 (predicate)	svand_x	Logical AND
_mm_or_si128 (arithmetic)	svorr_z	Bit-wise OR
_mm_or_si128 (predicate)	svorr_x	Logical OR
_mm_xor_si128 (arithmetic)	sveor_x	Bit-wise XOR
_mm_andnot_si128 (arithmetic)	svbic_x	Bit-wise ANDNOT
_mm_andnot_si128 (predicate)	svbic_z	Logical ANDNOT
_mm_cmpeq_epi{8,16}	svcmpeq	Equal comparison
_mm_cmpgt_epi{8,16}	svcmpgt	Greater than comparison
_mm_cmpge_epi16	svcmpge	Greater or equal comparison
_mm_load_si128	svld1	Memory load
_mm_store_si128	svst1	Memory store

A64FX

- First to implement SVE (512 bits)
- Present in Fugaku Supercomputer: Ranked first in the Top500 list from June 2020 to November 2021
- 48 cores
- 2.2 GHz

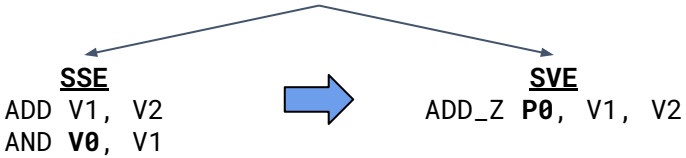


Optimizing BWA-MEM2

- General optimizations:
  - **Large pages**: We use 2MB pages instead of 4KB default pages
- BSW (extend) optimizations (already in GenArchBench):
  - **Predication**: We use SVE predication to substitute SSE masking

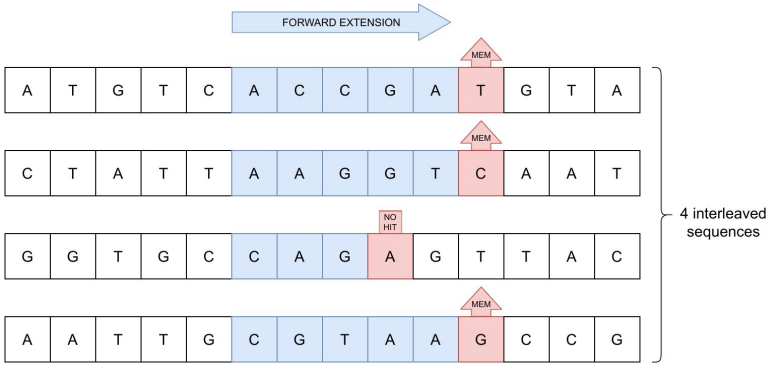
```
if (y < 0) x = y + z
else x = 0
```

```
V0/P0 = CMPLT V1, #0
```



Optimizing BWA-MEM2

- SMEM (seed) optimizations (already in GenArchBench):
  - **Inlining**: We force inline for one of the most used functions
  - **Popcnt**: We rewrite the population count operation (count number of 1 s in a vector) using SVE
  - **Interleaved sequences**: We interleave several sequences to hide memory latency



Methodology: Hardware Platforms

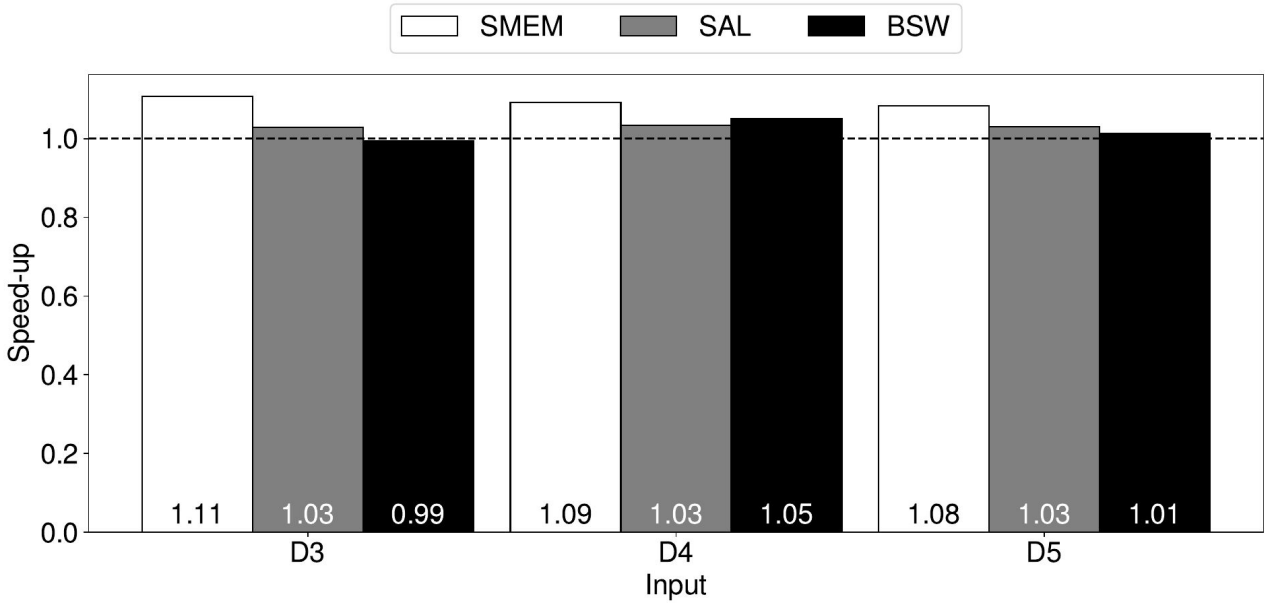
	A64FX	2 × Intel Xeon Platinum 8160 (SKX)
Cores	48	24 × 2
Frequency	2.2 GHz	2.1 GHz
Vector extension	SVE 512	AVX-512
Main memory	HBM2 32 GB & 1024 GB/s	DDR4 96 GB & 240 GB/s

Methodology: Inputs

We use *GRCh38* human genome as the reference: 3.05 gigabase-pairs

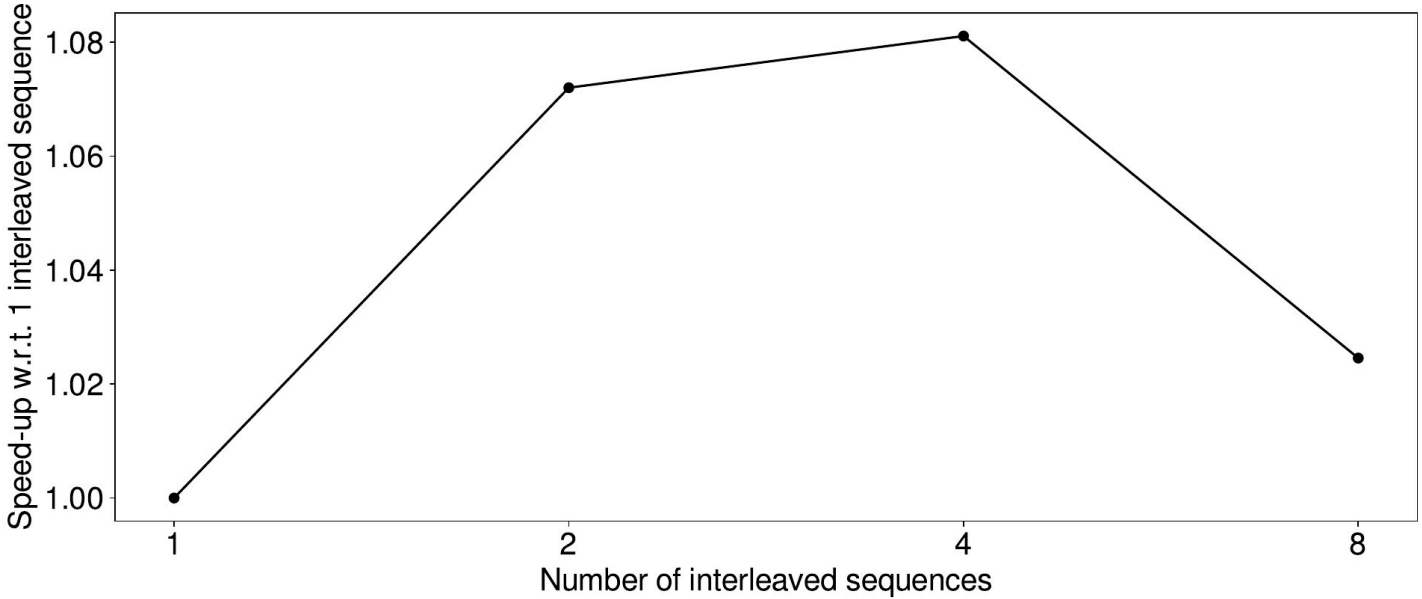
	D3	D4	D5
Organism	Homo Sapiens	Homo Sapiens	Homo Sapiens
Machine	Illumina Genome Analyzer II	Illumina HiSeq 2000	Illumina HiSeq 2000
Sequence length	76	101	101
Number of sequences	17.8 M	92.4 M	1,436.8 M
Run	SRR043348	SRR622461	SRR622457

Optimization Evaluation: Large Pages



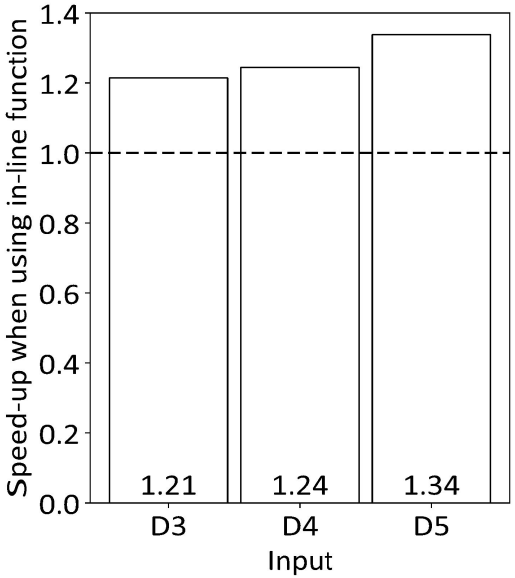
Larger pages benefits more the latency bound kernels

Optimization Evaluation: Interleaved Sequences

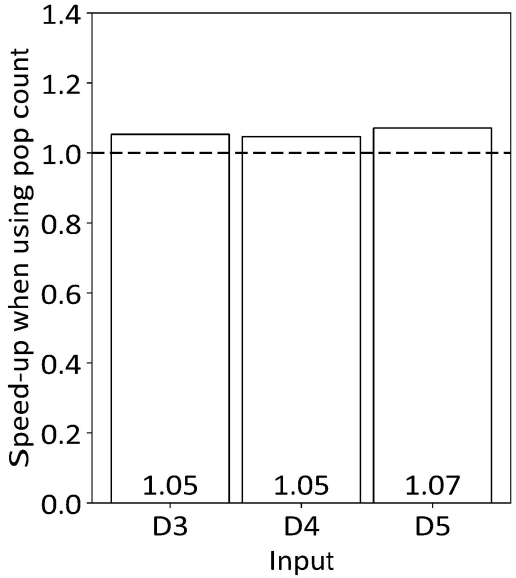


4 interleaved sequences speed up SMEM by 8%

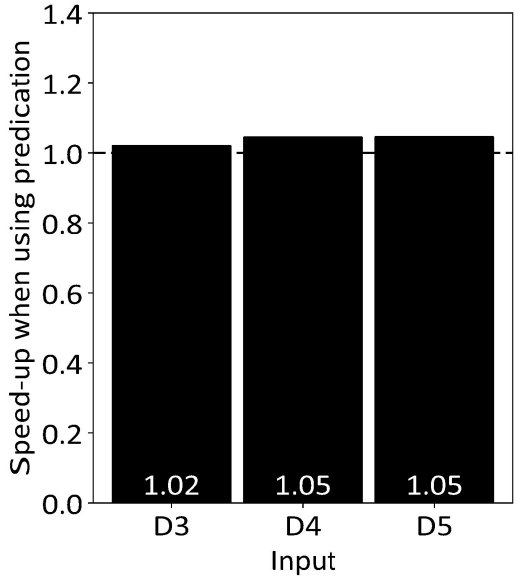
Optimization Evaluation



In-line (SMEM)

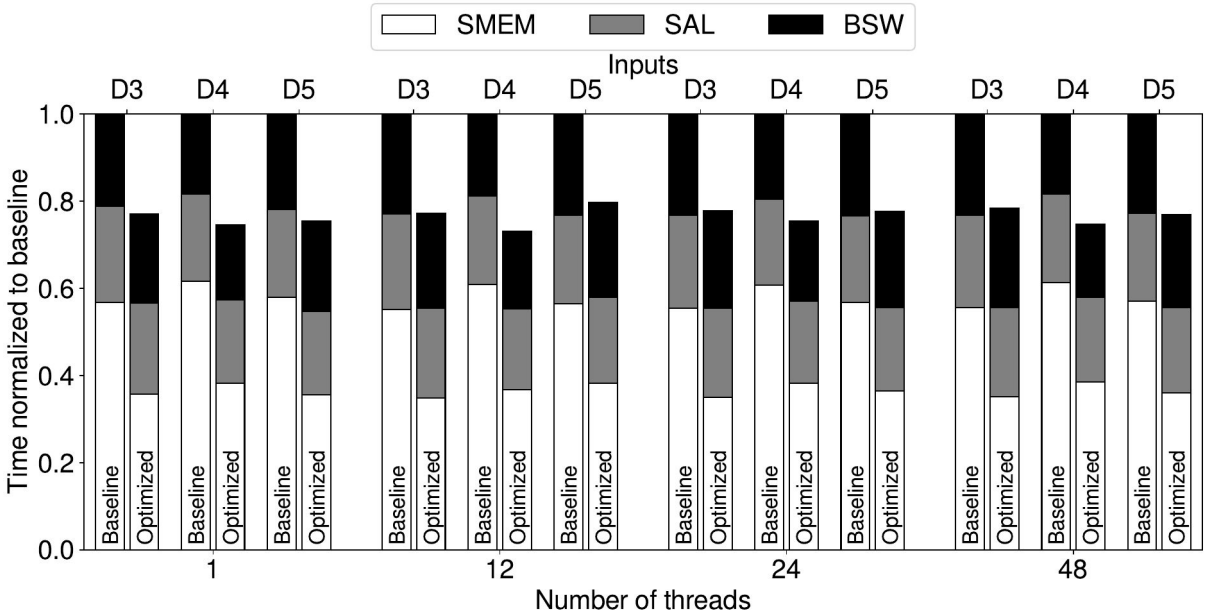


Population count (SMEM)



Predication (BSW)

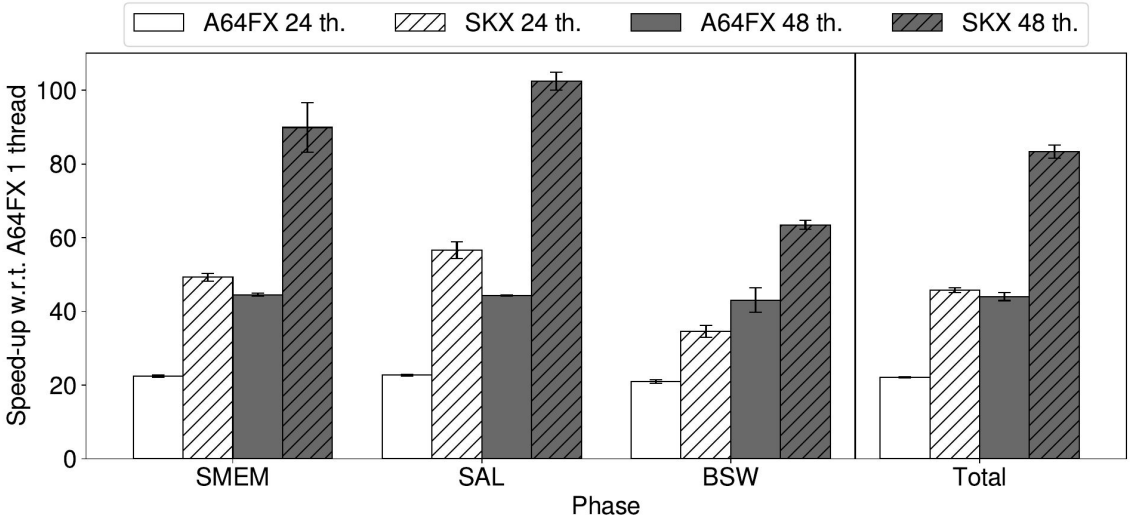
Final Version Evaluation



23.2% improvement on average for 48 threads

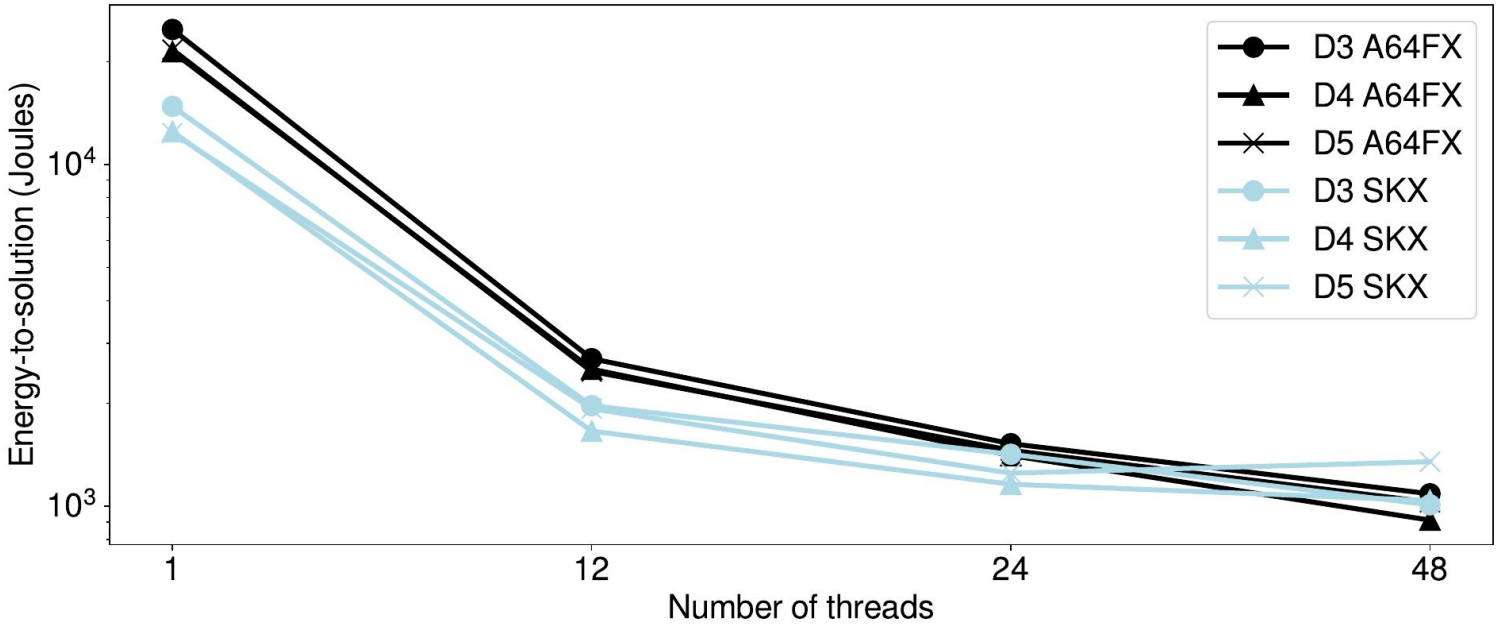
SMEM: 36.9% improvement on average for 48 threads

Evaluation: Comparison with Intel Skylake (SKX)



- Socket-to-socket (SKX 24 threads vs A64FX 48 threads):
  - SKX beats A64FX in latency bound kernels (less memory latency)
  - A64FX beats SKX in compute bound kernels (more memory bandwidth)
- SKX beats A64FX for the same number of threads (48 threads)

Evaluation: Comparison with SKX



A64FX beats SKX in terms of energy-to-solution by 11.6% on average for 48 threads

Porting and optimizing BWA-MEM2 conclusions

- We port BWA-MEM2 to ARM architecture
- We optimize the code for the A64FX: 23.2% of time reduction
- SKX system performs 1.89× better than the A64FX on average
- A64FX presents an energy-to-solution reduction of 11.6% w.r.t. SKX

https://gitlab.bsc.es/rlangari/bwa-a64fx

Rubén Langarita, Adrià Armejach, Pablo Ibáñez-Marín, Jesús Alastruey-Benedé, and Miquel Moretó (2023). *Porting and optimizing BWA-MEM2 using the Fujitsu A64FX processor*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 20(5), 3139-3153.

4 - Squire

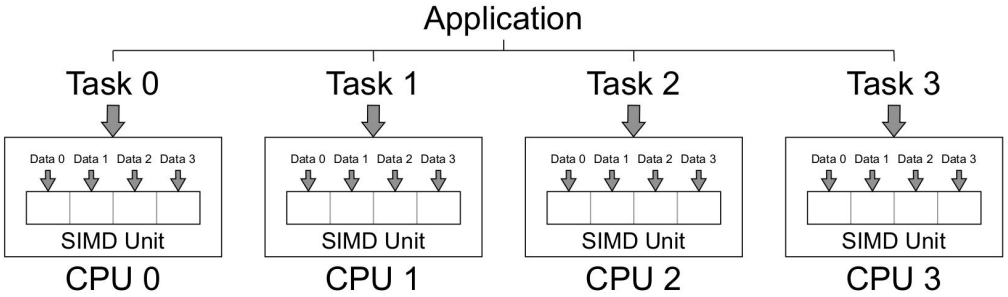
Hardware Accelerator for Dependency-Bound Kernels

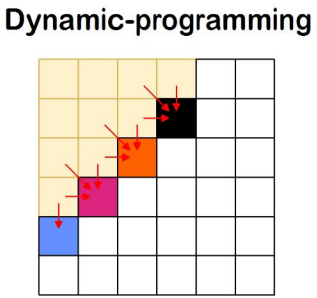
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Fine-Grain Parallelism on Dependency-Bound Kernels

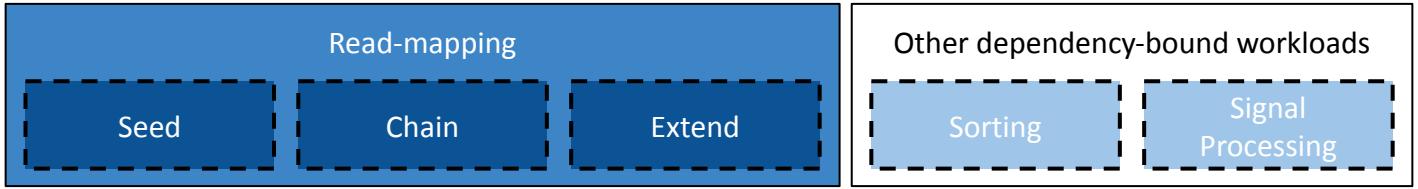
- Coarse-grain parallelism → CPUs (e.g. sequences)
- Fine-grain parallelism → SIMD (e.g. DP matrix)





- However, this kind of parallelism is challenging to exploit due to existing dependencies
- We need a general purpose hardware accelerator for dependency-bound kernels

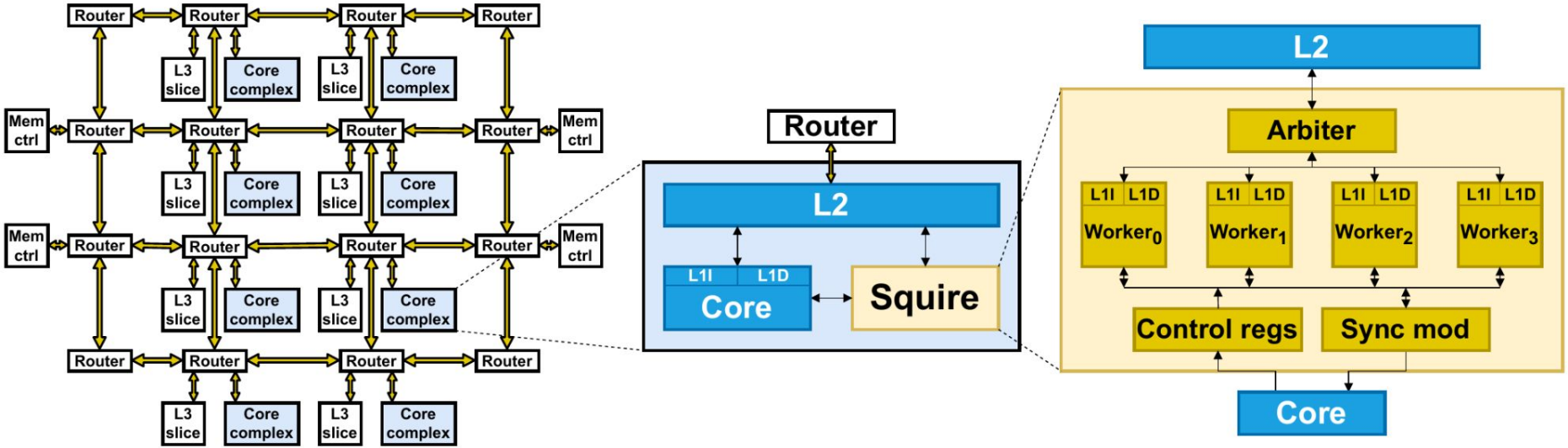
Dependency-Bound Use Cases: Beyond Genomics



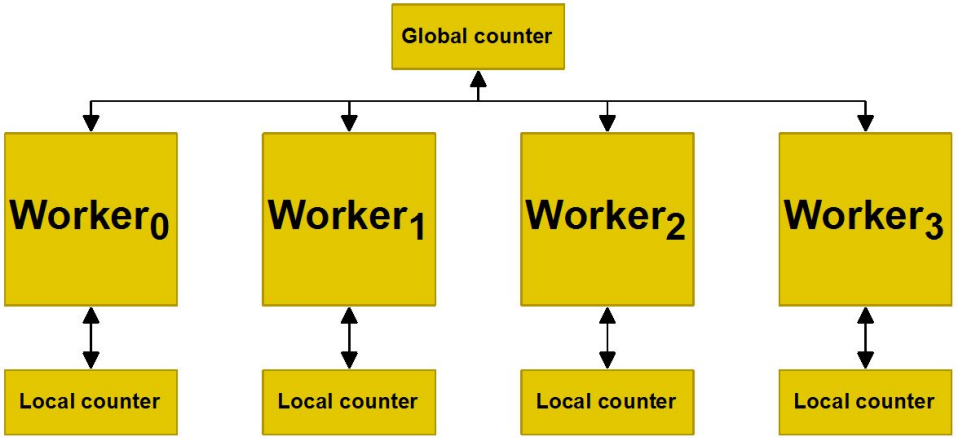
Squire is a hardware accelerator for dependency-bound kernels:

- Genomics: **Seed**, **Chain**, and **Smith-Waterman**
- Sorting: We use **radix sort** algorithm
- Signal processing: We use **Dynamic Time Warping** (DTW), an alignment algorithm for temporal sequences, such as video or audio

Squire


Router
L3 slice
Core complex
Mem ctrl
L2
L1I
L1D
Core
Squire
Arbiter
Worker0
Worker1
Worker2
Worker3
Control regs
Sync mod

Synchronization Module



Hardware counters that can be accessed in one cycle:

- **Local counters**: One per worker. Each worker can increment its local counter and read all the local counters.
- **Global counter**: Only one. All the workers can increment and read it.

Squire API Example: Radix Sort

```
function radix(A[N]):
    start_squire(radix_workers, A)
    wait_gcounter(num_workers())
    merge_sorted_arrays(A)

function radix_workers(A[N]):
    start = id_worker() × (N / num_workers())
    end = (id_worker() + 1) × (N / num_workers())
    radix_kernel(A[start:end])
    inc_gcounter()
    stop_worker()
```

Initiate Squire

Meanwhile, the main core is waiting the workers

And merges the sorted arrays

Workers start executing this function

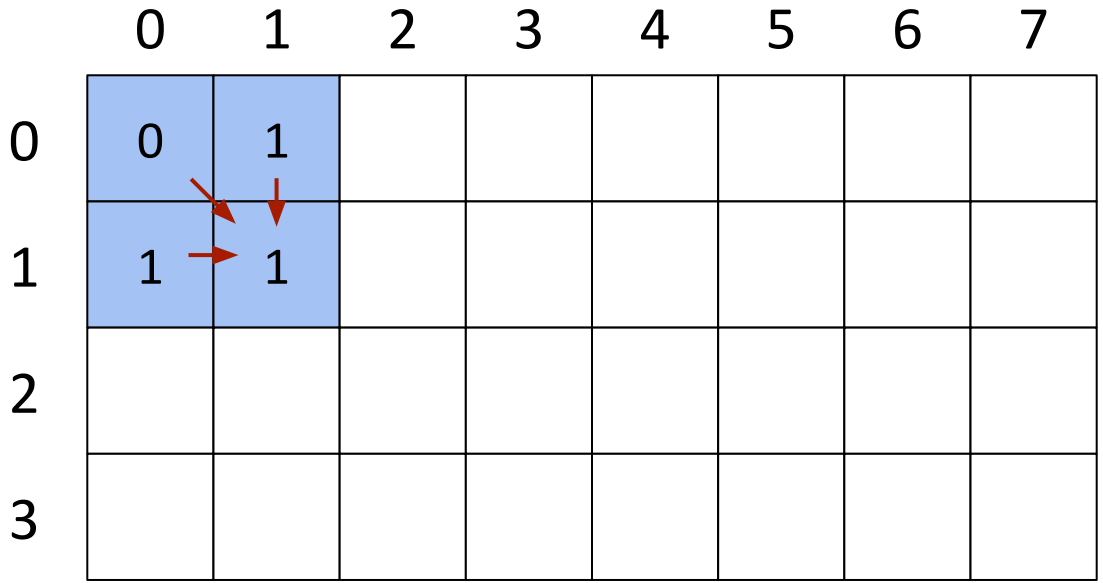
Each worker takes a chunk of the array

Each worker sorts its chunk

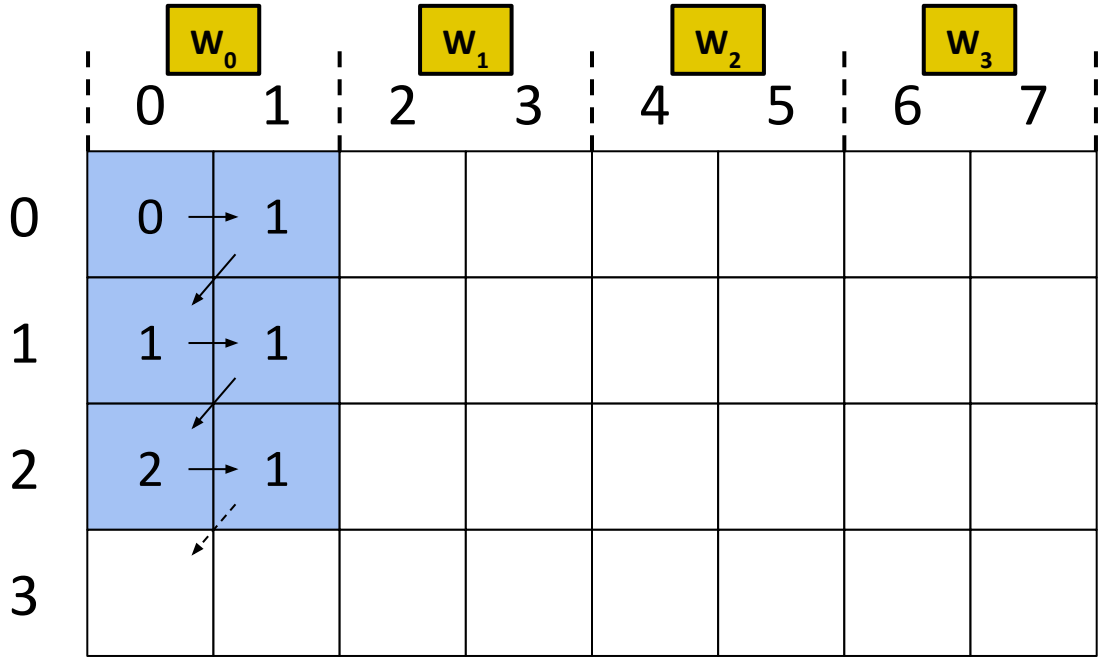
Increments the global counter

And stops

Squire Execution Example: 2D DP Matrix (SW/DTW)



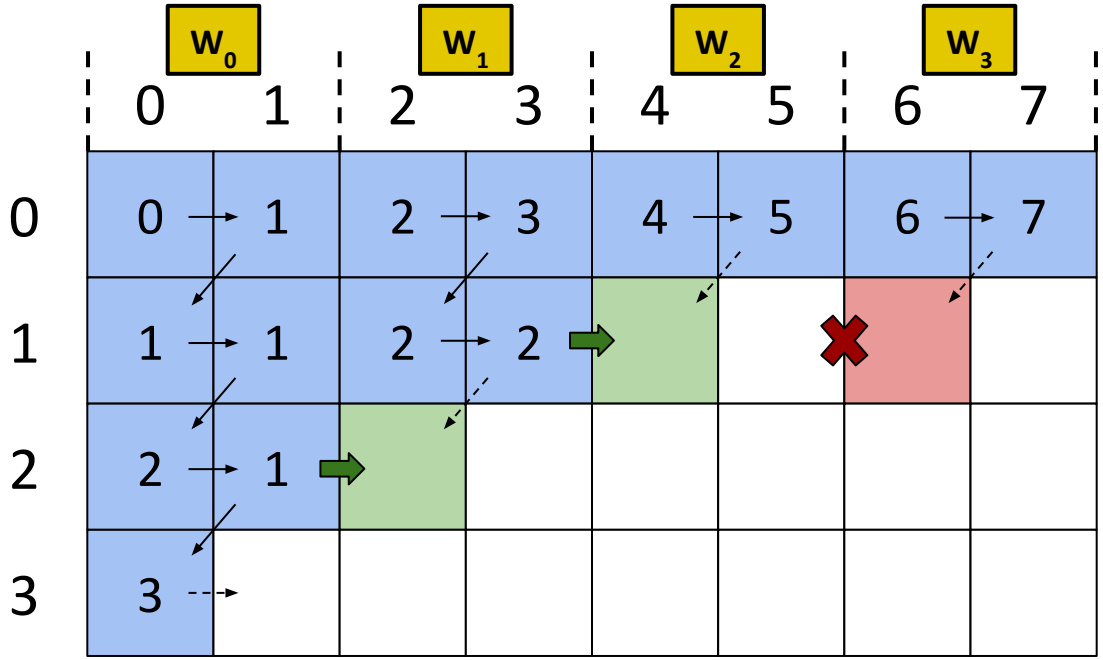
Squire Execution Example: 2D DP Matrix (SW/DTW)



Local Counters

0	1	2	3
3	0	0	0

Squire Execution Example: 2D DP Matrix (SW/DTW)



Local Counters

0	1	2	3
3	2	1	1

Methodology

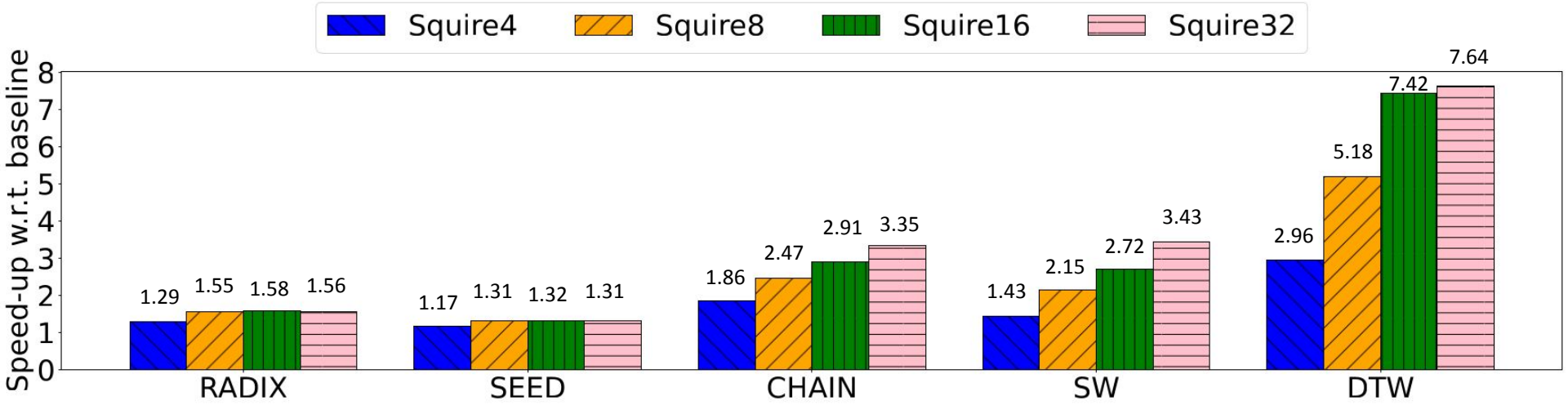
We use **Gem5** full system simulator to model Squire

Cores	8 Neoverse-N1-like Armv8 out-of-order cores 2.4 GHz **each core is equipped with one Squire**
Coherence protocol	MOESI-like AMBA 5 CHI specification
Network topology	4×4 2D mesh, 1 cycle routers, 1 cycle links
Memory	1 HBM2 stack, 300 GB/s
Squire	Each Squire consists of several workers, the synchronization module, the control registers, and the arbiter
Worker	Cortex-M35P-like Armv8 4-stage dual-issue in-order cores 2.4 GHz

Methodology

- Speed-up w.r.t. not using Squire, i.e., the CPU computes the fine-grain parallel workloads
- We evaluate 5 workloads:
  - Radix sort (**RADIX**)
  - Seeding stage from Minimap2 (**SEED**)
  - Chain algorithm from Minimap2 (**CHAIN**)
  - Smith-Waterman (**SW**)
  - Dynamic Time Warping (**DTW**)
- We evaluate an end-to-end application: **read-mapping tool**

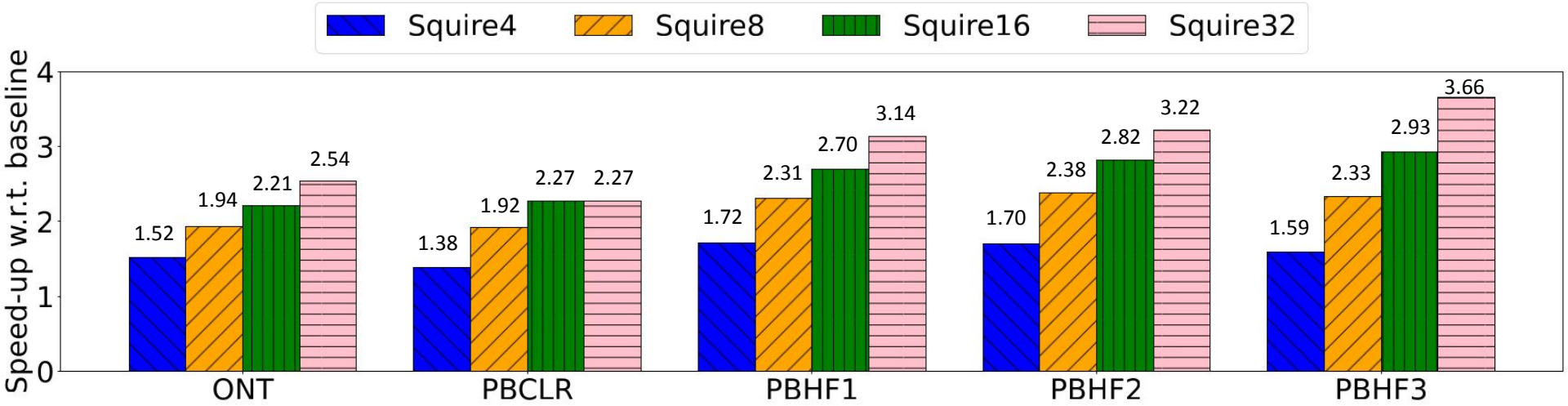
Evaluation



Speed-ups of up to 7.64× (32 workers)

We advocate for a balanced design of **16 workers**: average speed-up of 3.19× and up to 7.42×

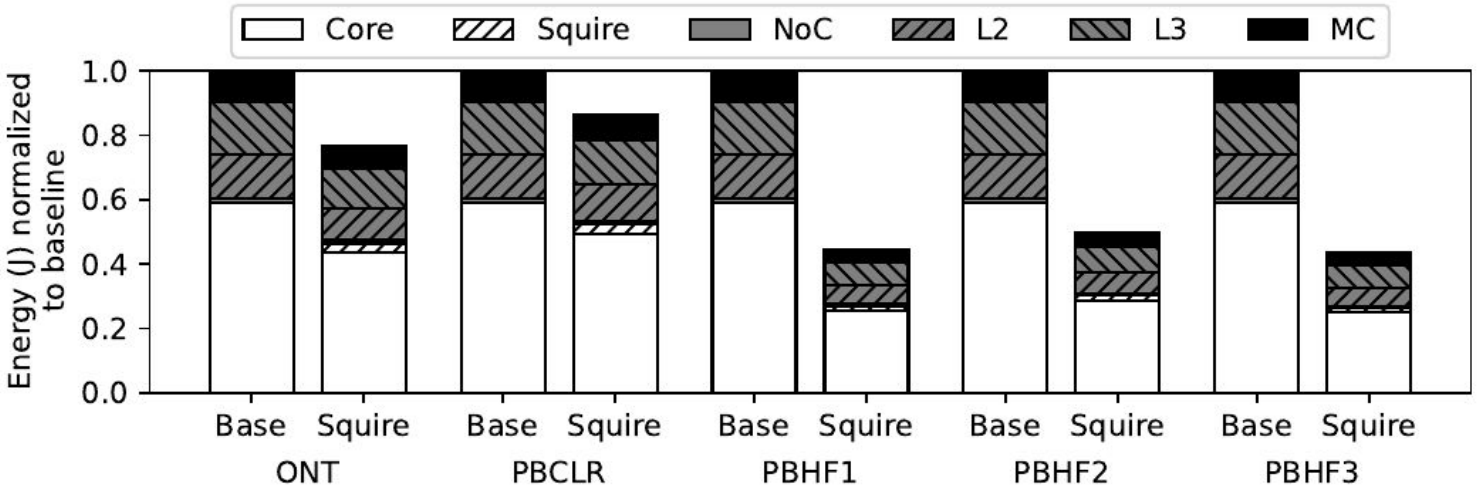
Evaluation: End-to-End Application



Speed-ups of up to 3.66× (32 workers)

For **16 workers**: average speed-up of 2.59× and up to 2.93×

Evaluation: Energy / Area



Up to 56% energy consumption reduction

Out-of-Order (Arm Neoverse-N1) has an area overhead of 10.5% for a 16-worker Squire

Squire conclusions

- Squire: general-purpose accelerator for dependency-bound fine-grain parallelism
- We advocate for a **16-workers** configuration:
  - DP kernels: average speed-up of 3.19× and up to 7.42×
  - End-to-end application: average speed-up of 2.59× and up to 2.93×
  - Energy reduction of up to 56%
  - Area overhead of 10.5% w.r.t. OoO core

Rubén Langarita, Adrià Armejach, Santiago Marco-Sola, Pablo Ibáñez-Marín, Jesús Alastruey-Benedé, and Miquel Moretó (2025). *Squire: A General-Purpose Accelerator to Exploit Fine-Grain Parallelism on Dependency-Bound Kernels*. Parallel Architectures and Compilation Techniques 2025.

Conclusions

Conclusions

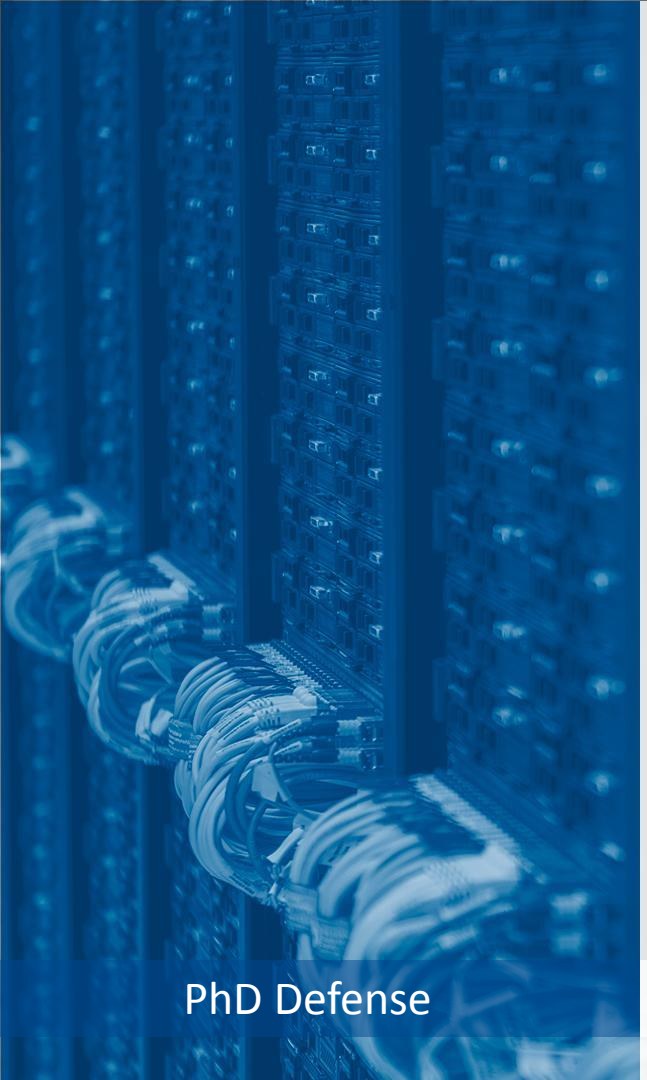
- **GenArchBench**: Genomics benchmark suite for ARM HPC systems
  - 13 kernels from the most widely-used genomics tools
  - GenArchBench enables comparing different systems and ISAs in a standardized manner
  - We evaluate GenArchBench in 2 ARM machines and 2 x86 machines
- **COFI**: FM-Index compression for large k-steps
  - We find redundant data when increasing the k-step in the state-of-the-art FM-Index
  - COFI is the first proposal to enable 15-step FM-Index with reasonable memory footprint
  - Speed-up of up to 2.14× w.r.t. state-of-the-art
- **BWA-MEM2 port** to ARM architecture
  - Optimized in A64FX (first processor to implement SVE) processor: 23.2% of time reduction
  - SKX system performs 1.89× better than the A64FX on average
  - A64FX presents an energy-to-solution reduction of 11.6% w.r.t. SKX
- **Squire**: General-purpose accelerator for dependency-bound fine-grain parallelism
  - We expand our set of dependency-bound kernels beyond genomics: sorting and signal processing
  - Speed-up of up to 7.42×
  - Energy reduction of up to 56%
  - Area overhead of 10.5% per core

Future (currently doing) work

- Maintain and improve open source repositories
  - BWA-MEM2
  - Minimap2
  - GenArchBench
  - Bioconda
- Dynamic Programming Extension (DPE): SIMD ARM extension
  - We propose new SIMD instructions to tackle dynamic programming kernels: DTW, WFA and SW

Publications

- **Rubén Langarita**, Adrià Armejach, Javier Setoain, Pablo Ibáñez-Marín, Jesús Alastruey-Benedé, and Miquel Moretó (2020). *Compressed sparse FM-index: Fast sequence alignment using large K-steps*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 19(1), 355-368.
- **Rubén Langarita**, Adrià Armejach, Pablo Ibáñez-Marín, Jesús Alastruey-Benedé, and Miquel Moretó (2023). *Porting and optimizing BWA-MEM2 using the Fujitsu A64FX processor*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 20(5), 3139-3153.
- Lorién López-Villellas, **Rubén Langarita**, Asaf Badouh, Víctor Soria-Pardos, Quim Aguado-Puig, Guillem López-Paradís, Max Doblas, Javier Setoain, Chulho Kim, Makoto Ono, Adrià Armejach, Santiago Marco-Sola, Jesús Alastruey-Benedé, Pablo Ibáñez, and Miquel Moretó (2024). *GenArchBench: A genomics benchmark suite for arm HPC processors*. Future Generation Computer Systems, 157, 313-329.
- **Rubén Langarita**, Adrià Armejach, Santiago Marco-Sola, Pablo Ibáñez-Marín, Jesús Alastruey-Benedé, and Miquel Moretó (2025). *Squire: A General-Purpose Accelerator to Exploit Fine-Grain Parallelism on Dependency-Bound Kernels*. Parallel Architectures and Compilation Techniques 2025.



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Universidad Zaragoza

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Improving Performance of Genomics Workloads through Software Optimizations and Hardware Acceleration

Rubén Langarita Benítez

Universitat Politècnica de Catalunya

Barcelona Supercomputing Center

PhD Defense

Adrià Armejach (BSC/UPC) **Jesús Alastruey Benedé** (Unizar)

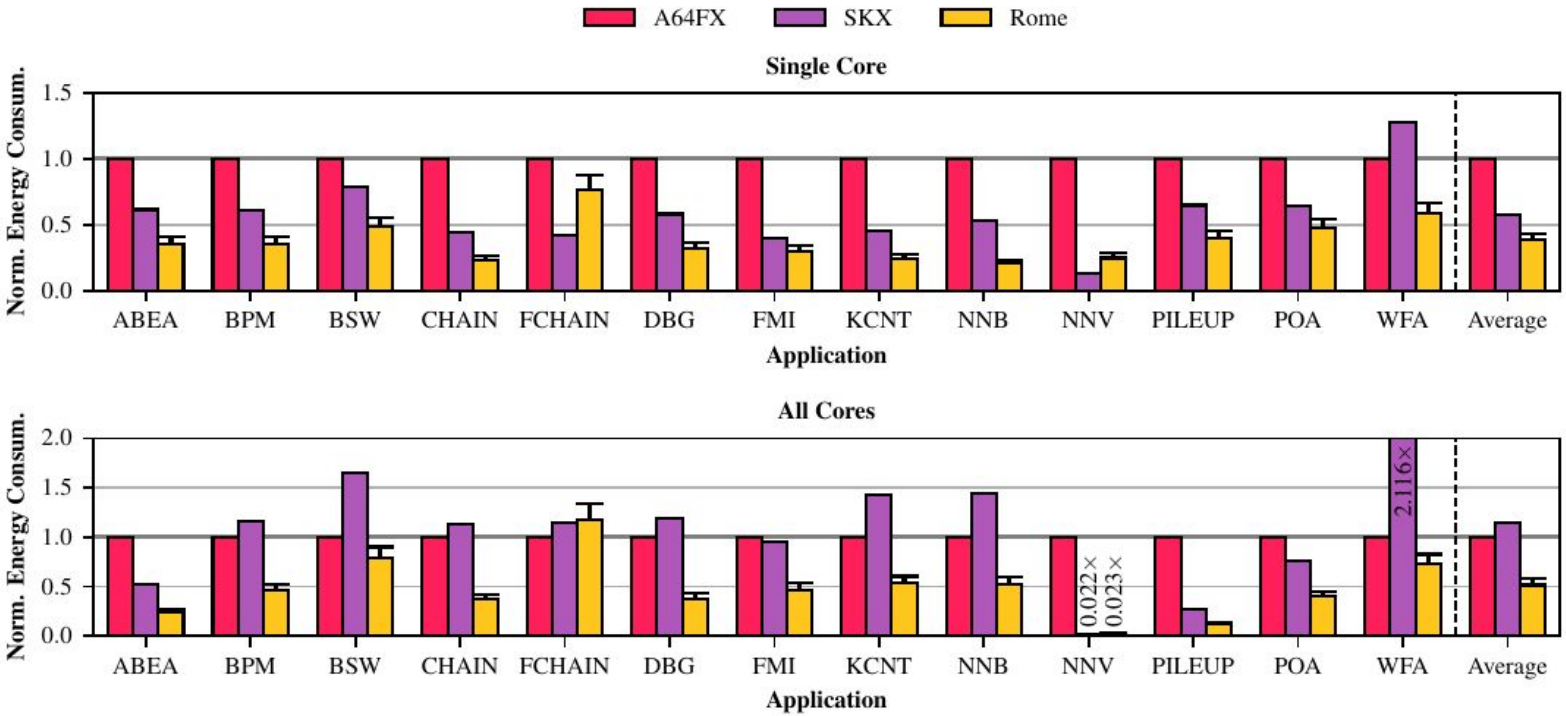
Back-Up

GenArchBench

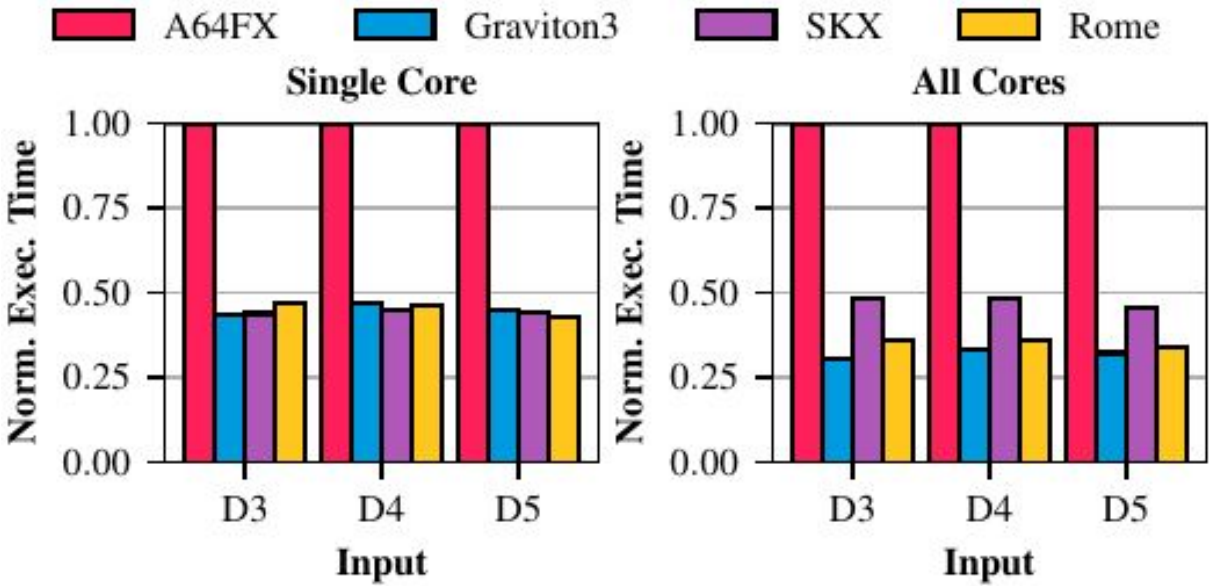
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

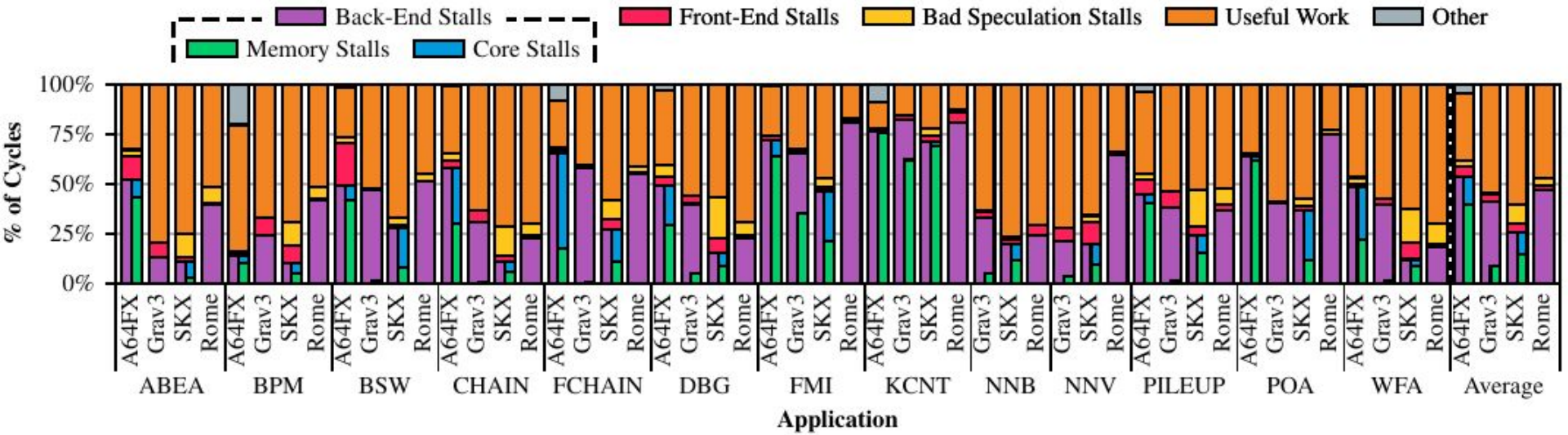
GenArchBench Energy



GenArchBench BWA-MEM2

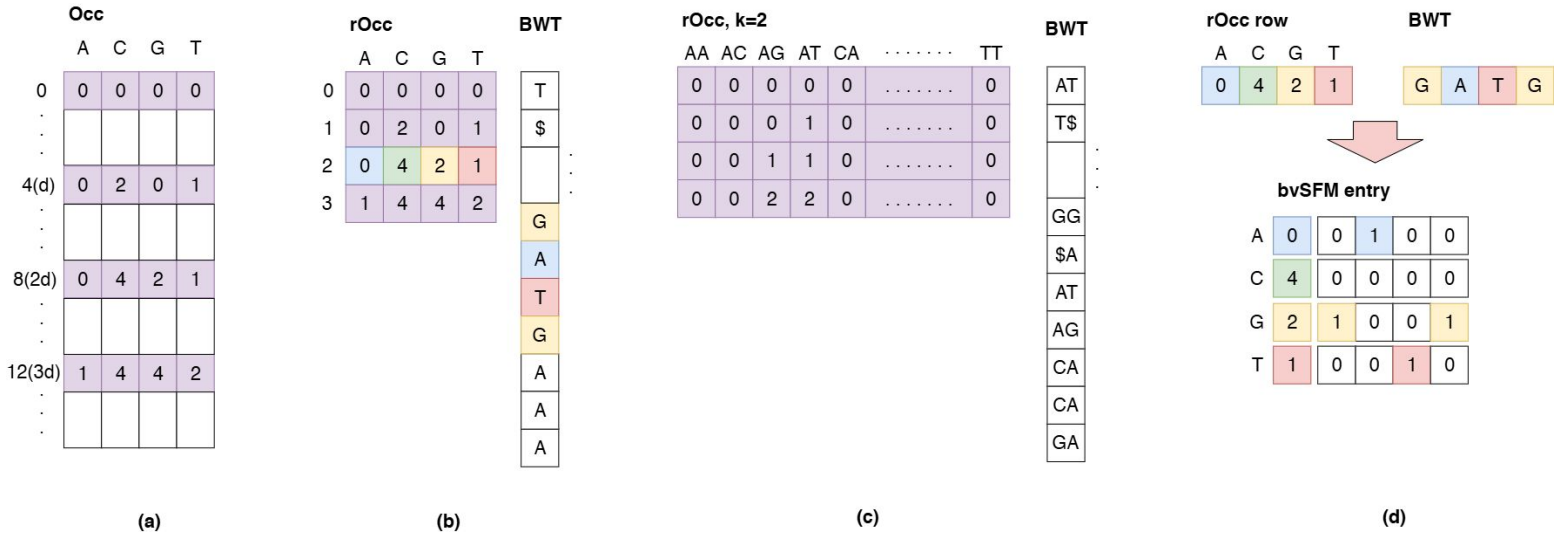


GenArchBench Stalls



COFI

State-of-the-art FM-Index



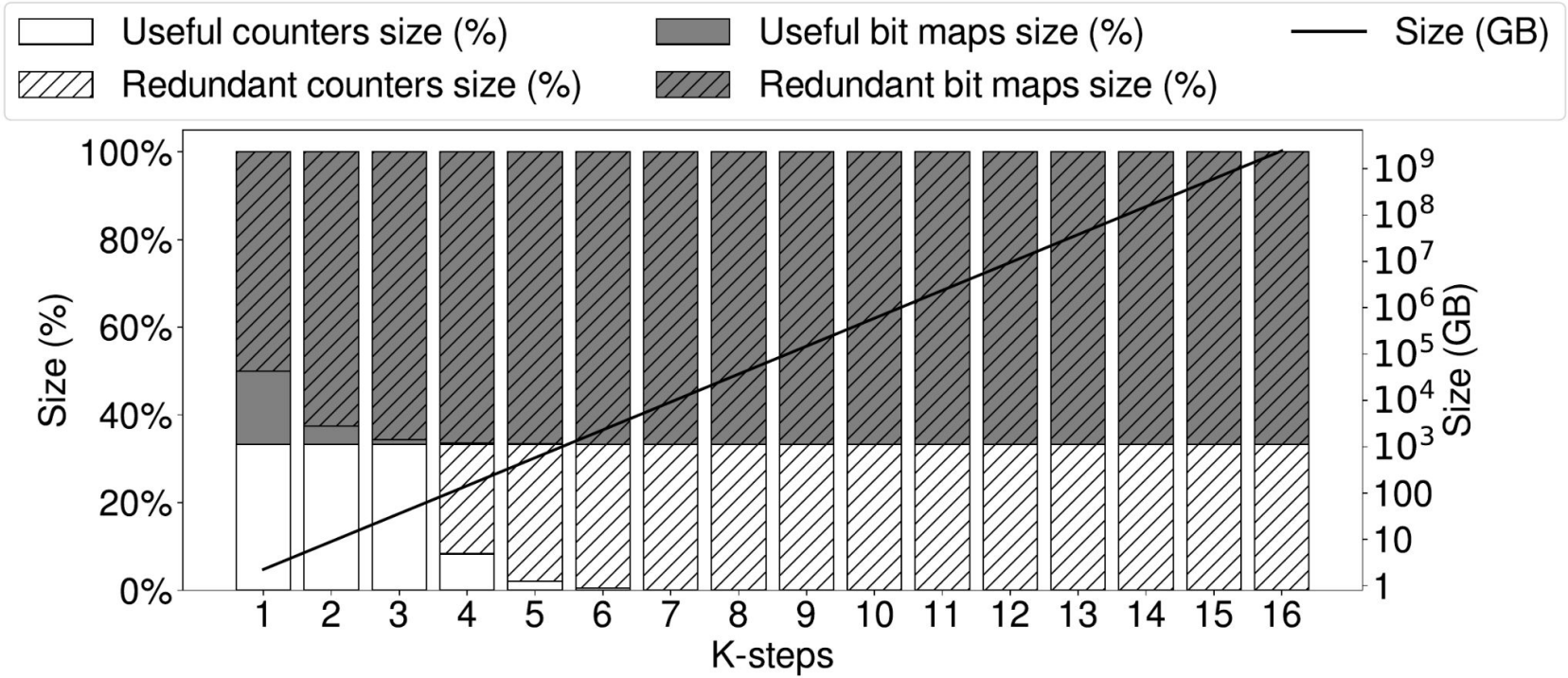
a) Standard FM-Index

b) Sampled FM-Index: Store 1 out of 4 Occ rows

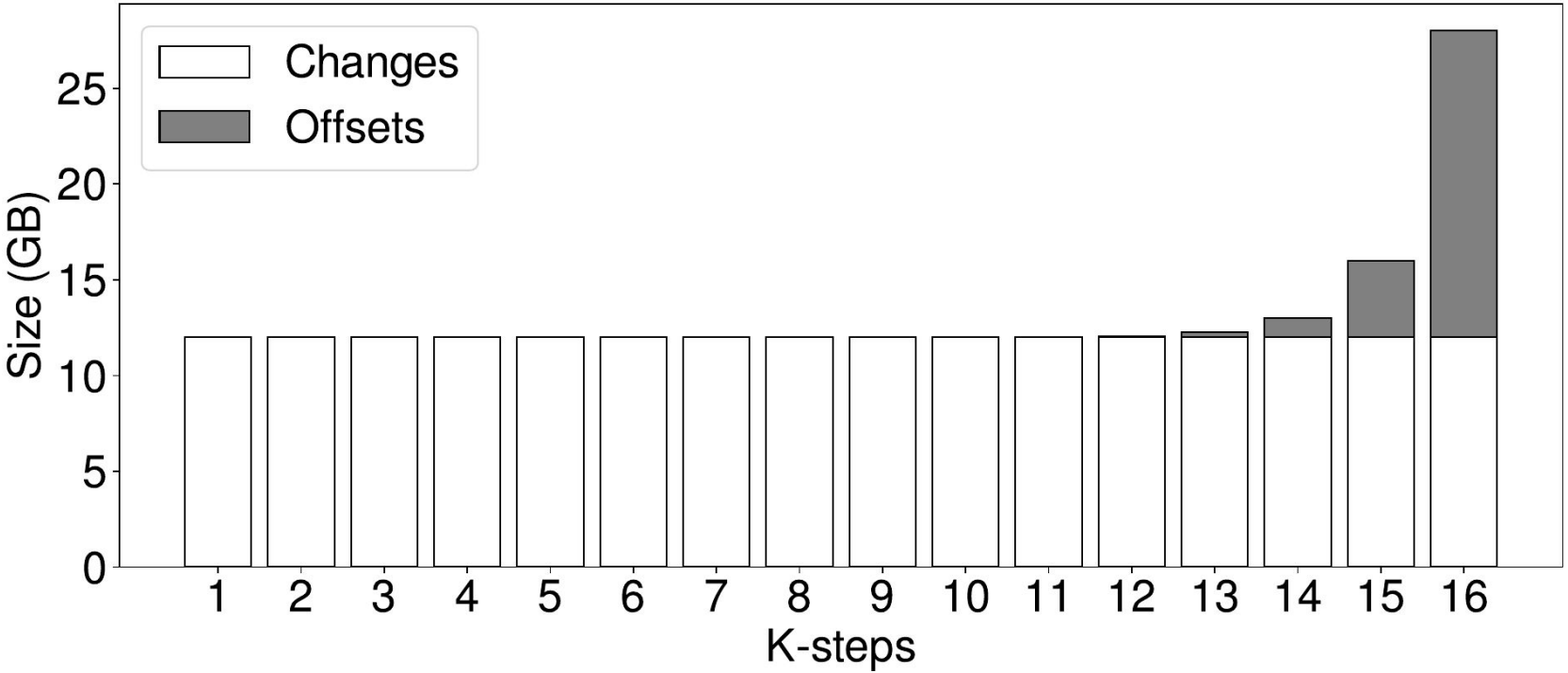
c) K-step FM-Index: Increase alphabet size, and the \textbf{bases searched per iteration}

d) bit-vector Sampled FM-Index (bvSFM): Use population count to count the number of base-pairs in BWT

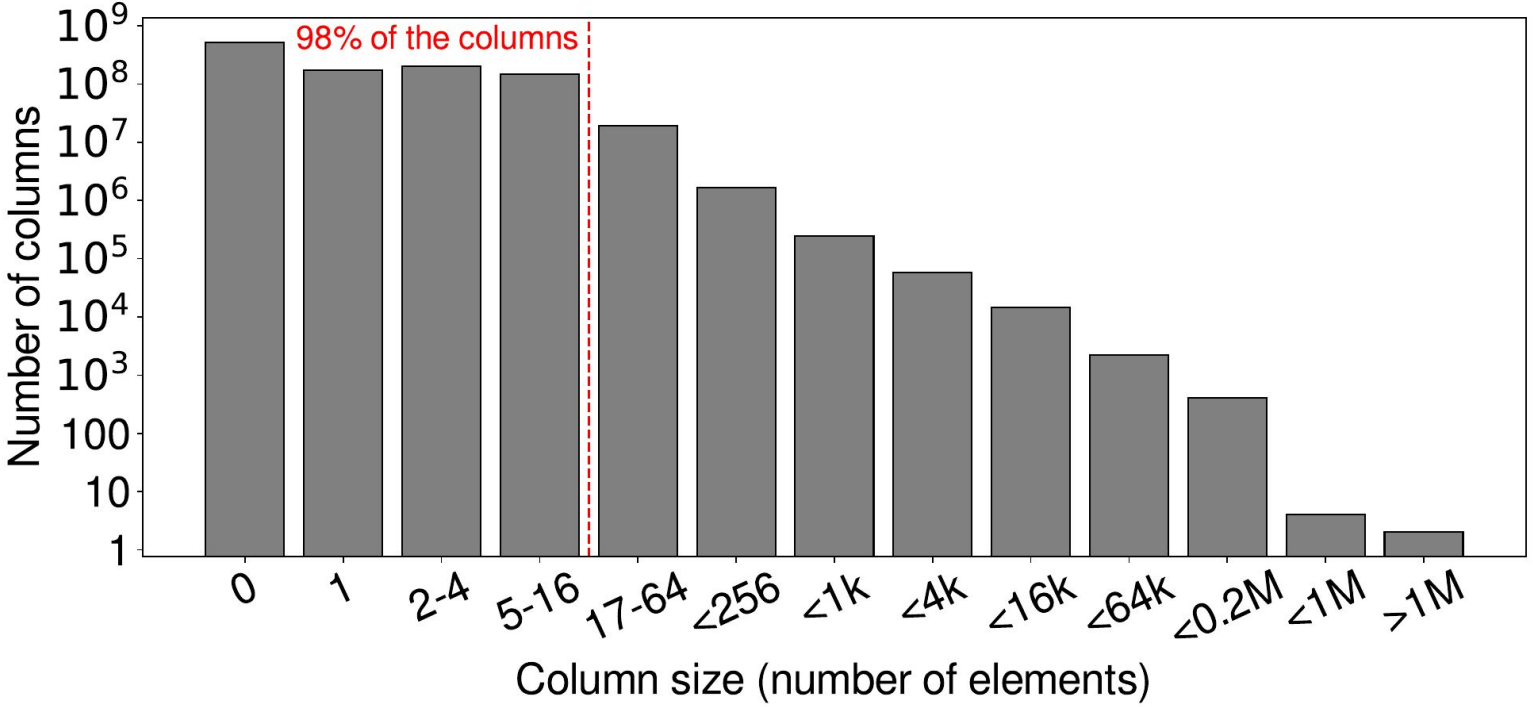
COFI Baseline Memory



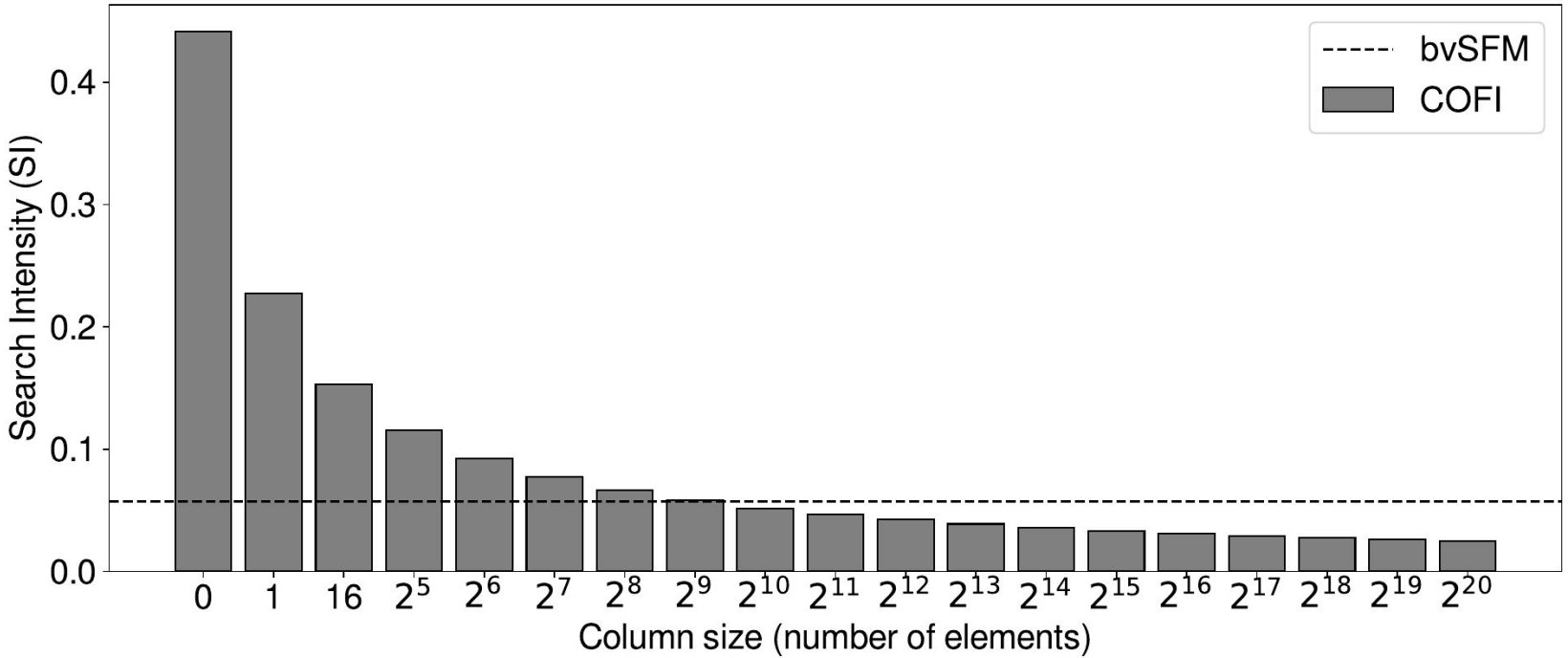
COFI Memory


Changes
Offsets
Size (GB)
0
5
10
15
20
25
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
K-steps

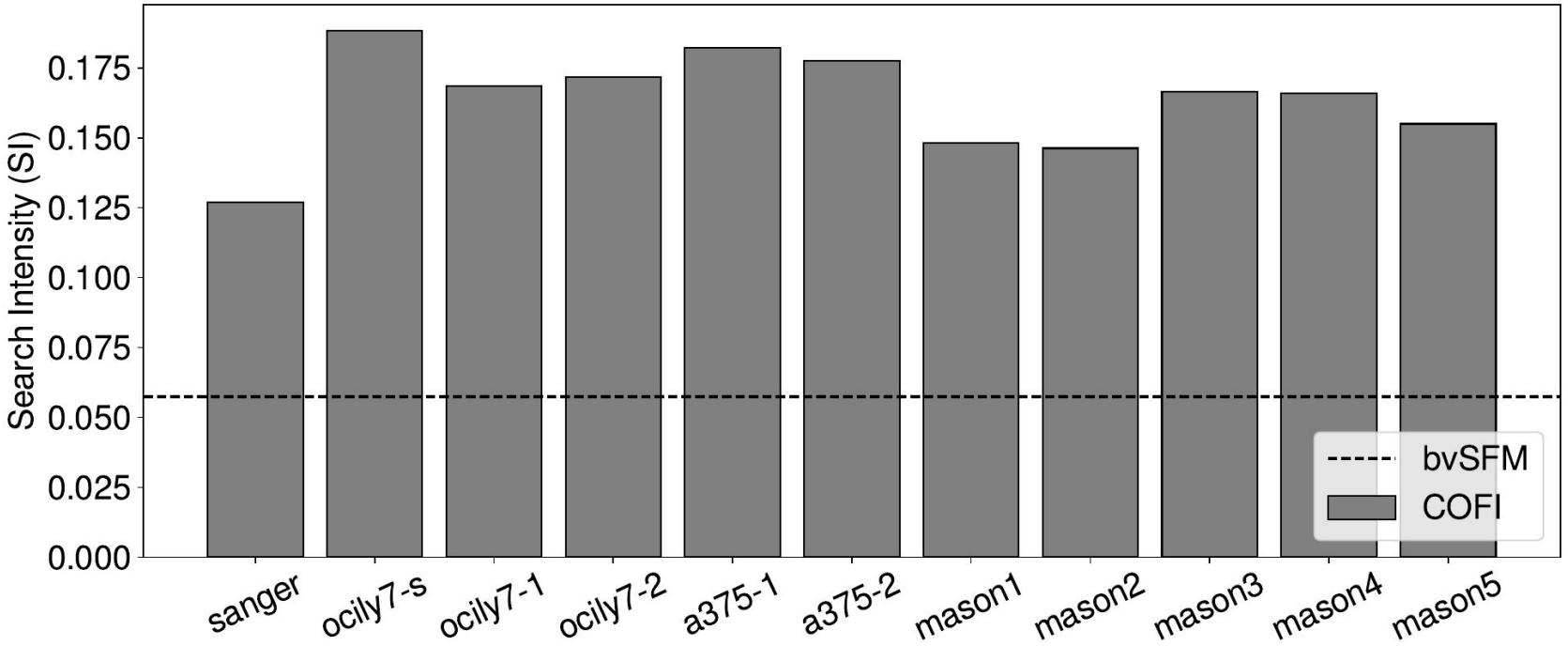
COFI Column Size



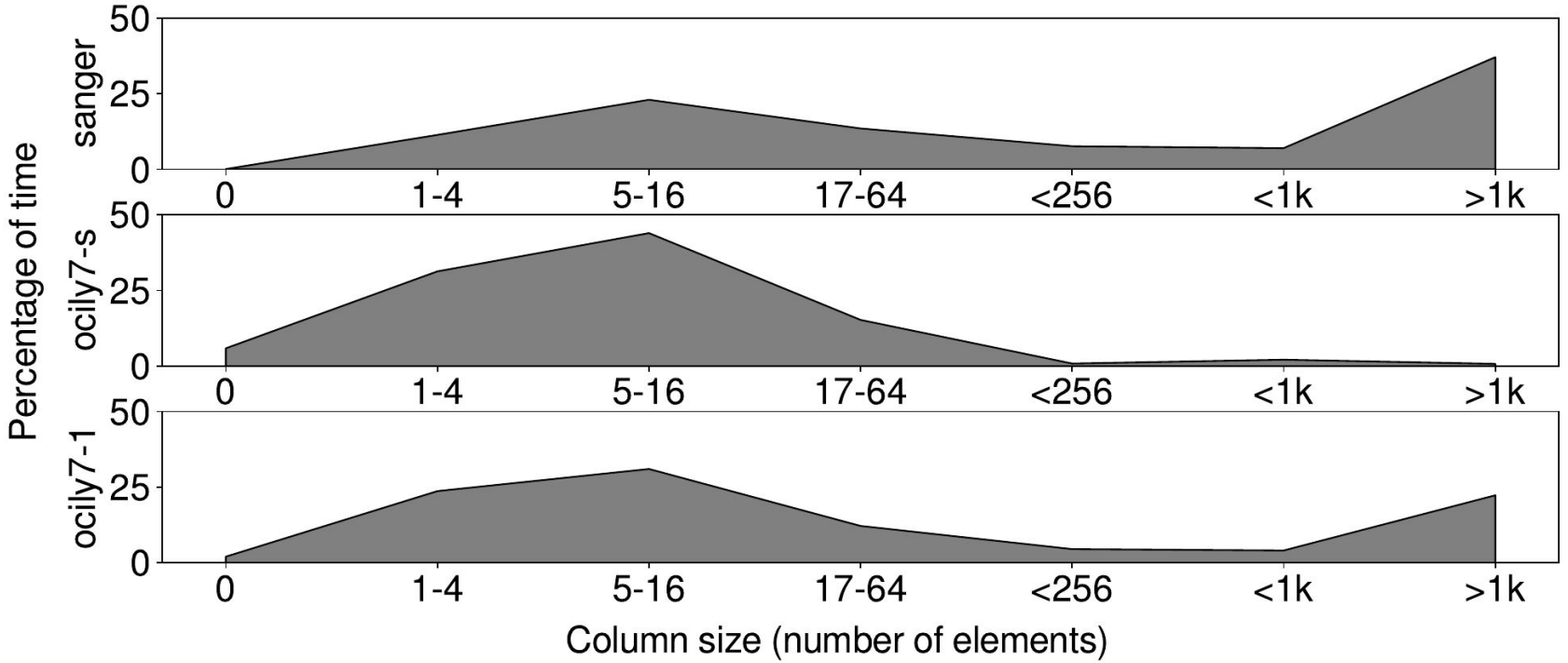
COFI SI Columns



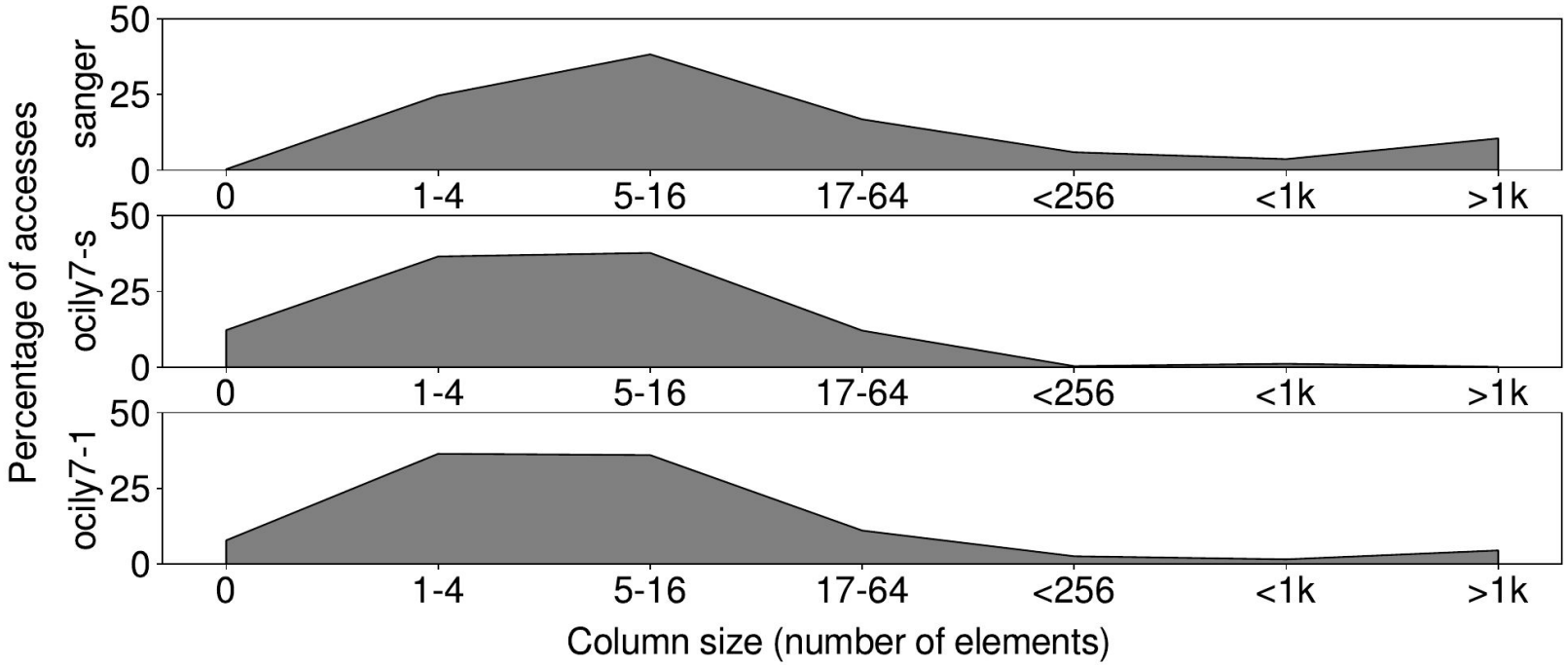
COFI SI Inputs



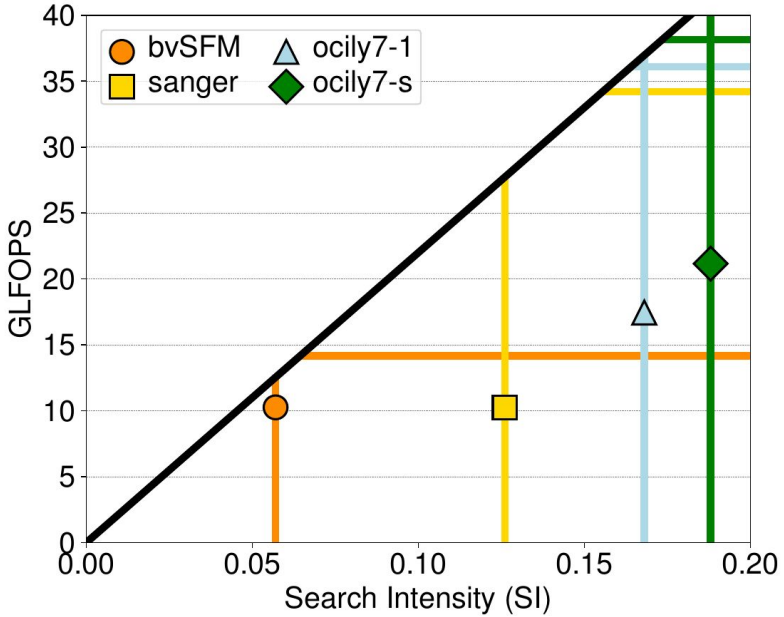
COFI Times Columns



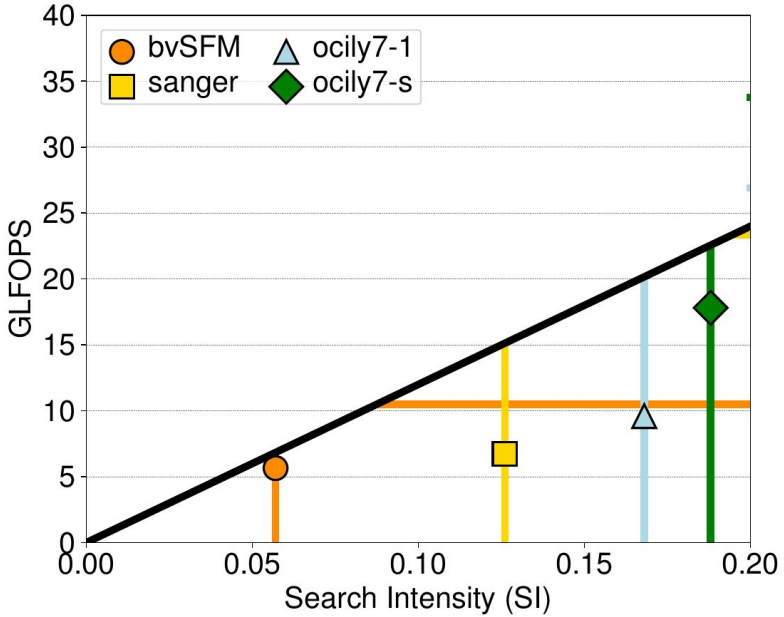
COFI Occurrences Columns



COFI Roofline

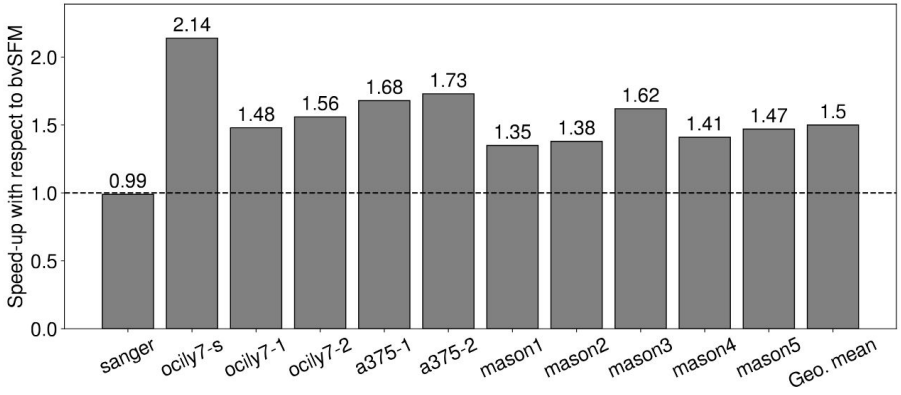


KNL

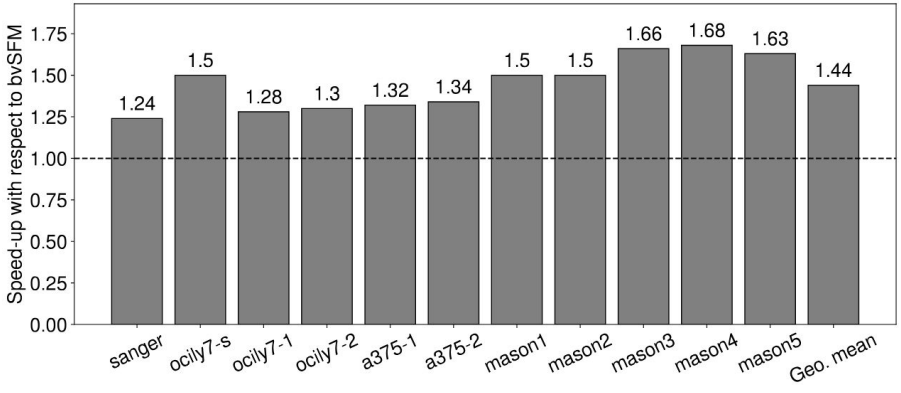


SKX

COFI vs SoA GRCh37



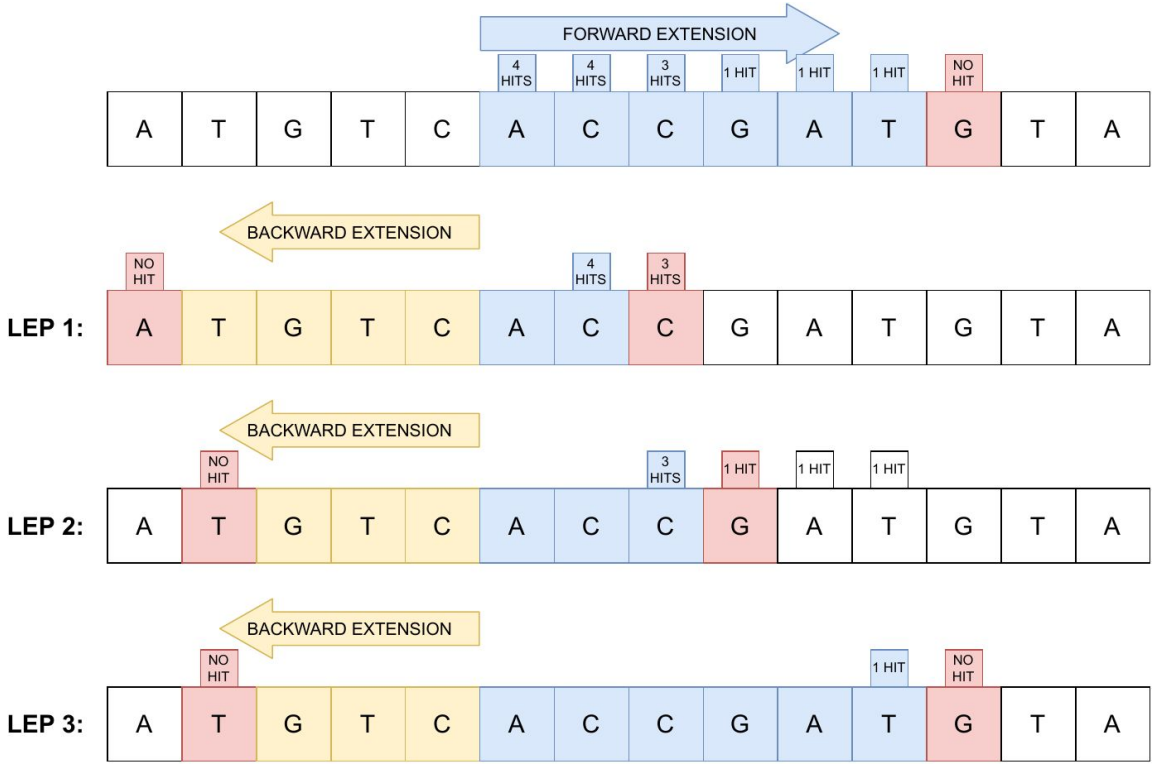
KNL



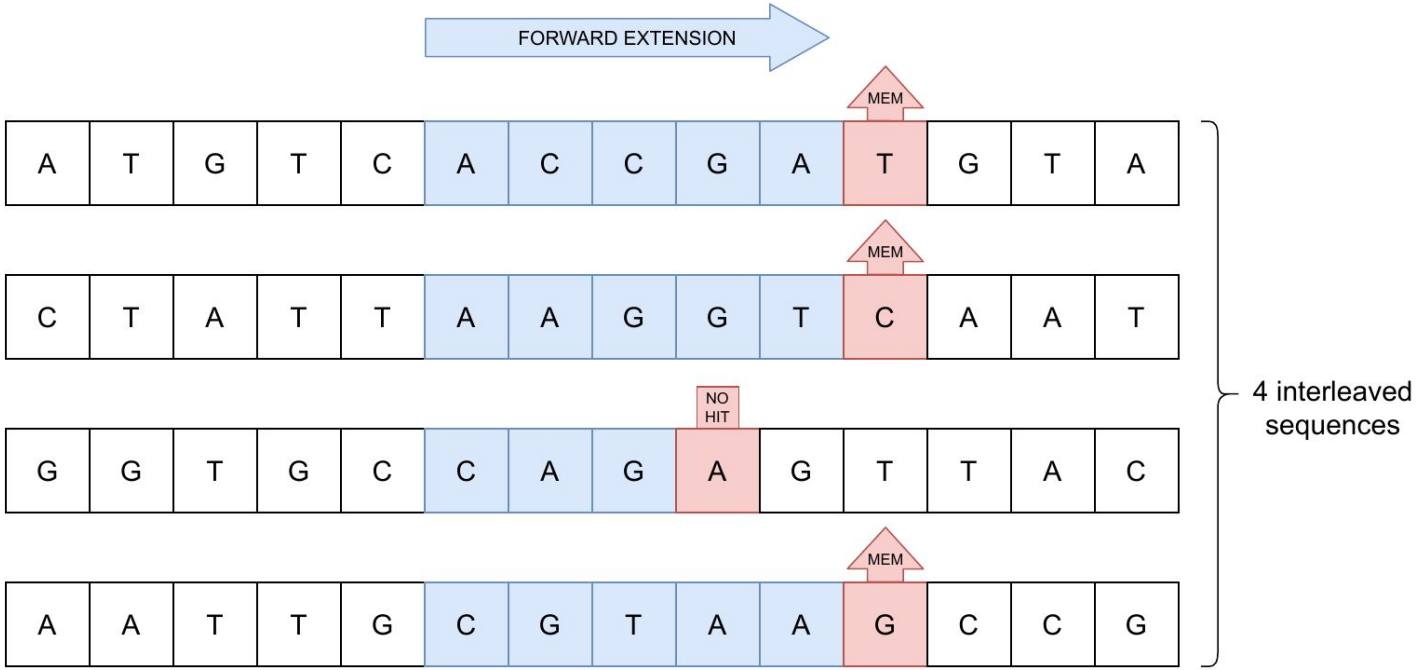
SKX

BWA-MEM2

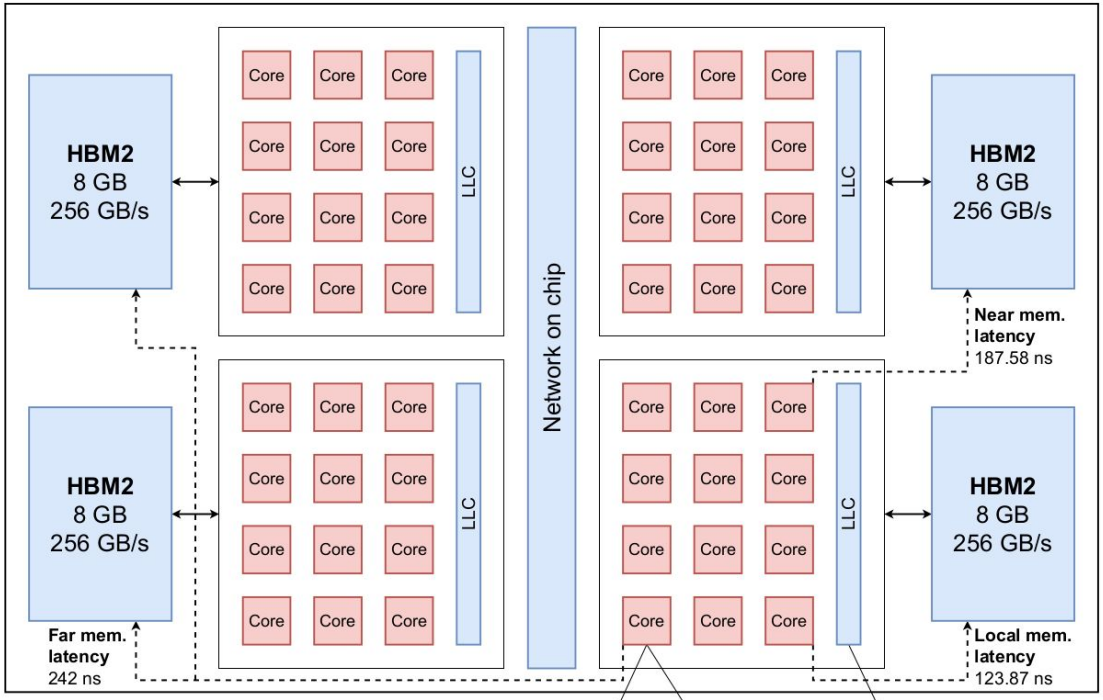
SMEM


FORWARD EXTENSION
4 HITS
4 HITS
3 HITS
1 HIT
1 HIT
1 HIT
NO HIT
A T G T C A C C G A T G T A
BACKWARD EXTENSION
NO HIT
4 HITS
3 HITS
LEP 1:
A T G T C A C C G A T G T A
BACKWARD EXTENSION
NO HIT
3 HITS
1 HIT
1 HIT
1 HIT
LEP 2:
A T G T C A C C G A T G T A
BACKWARD EXTENSION
NO HIT
1 HIT
NO HIT
LEP 3:
A T G T C A C C G A T G T A

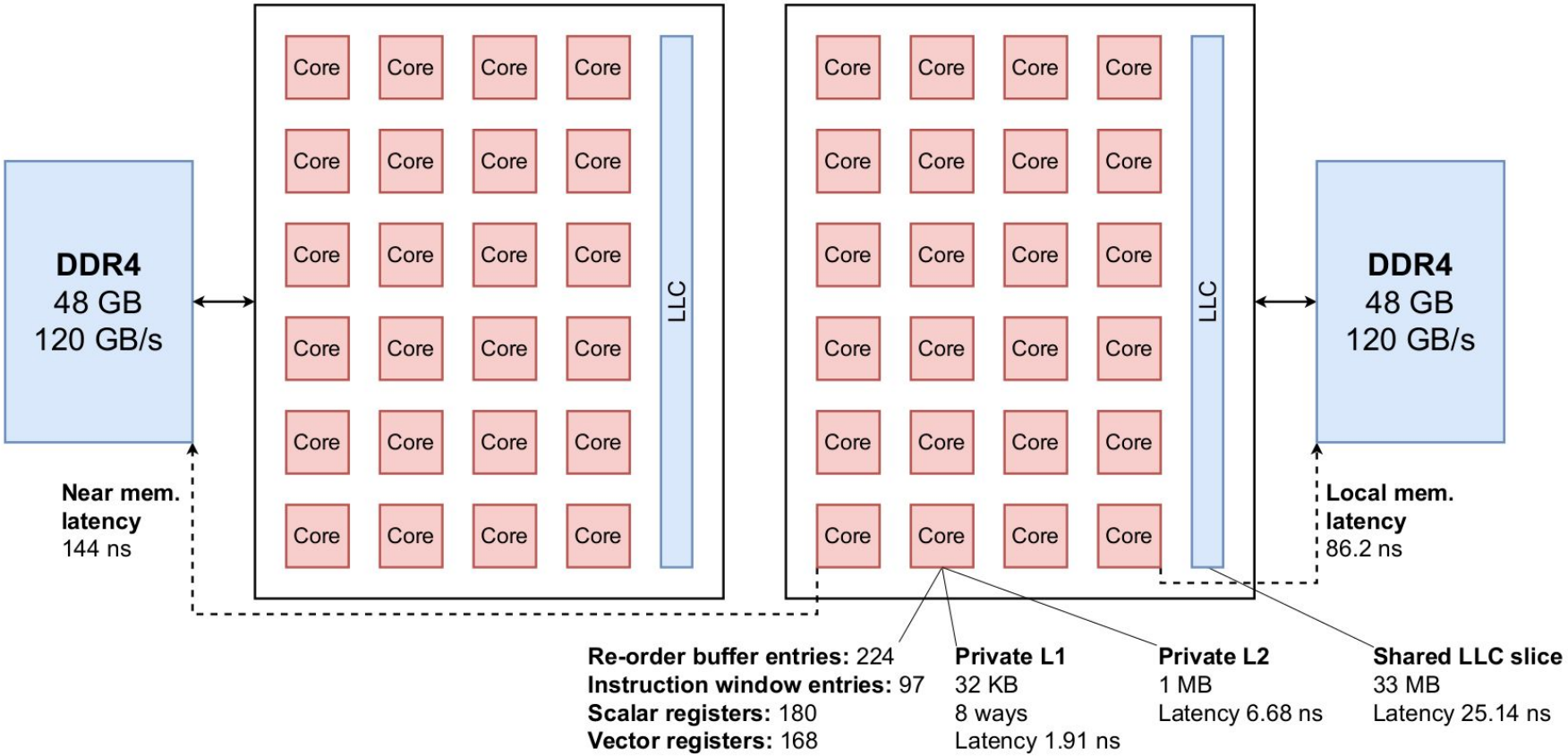
Interleaving



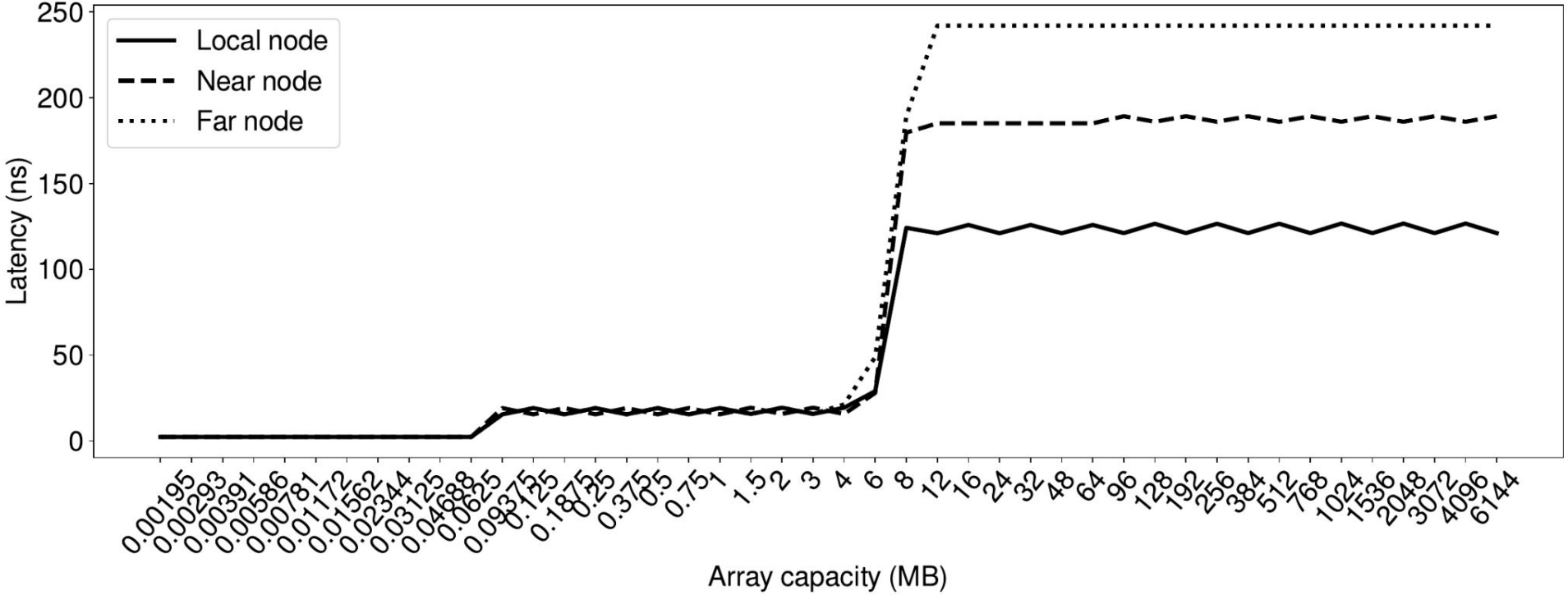
A64FX


HBM2
8 GB
256 GB/s
HBM2
8 GB
256 GB/s
HBM2
8 GB
256 GB/s
HBM2
8 GB
256 GB/s
Core
LLC
Network on chip
Near mem.
latency
187.58 ns
Far mem.
latency
242 ns
Local mem.
latency
123.87 ns
Re-order buffer entries: 128
Instruction window entries: 79
-20x2 arithmetic
-10x2 memory
-19 branch
Scalar registers: 96
Vector registers: 128
Private L1
64 KB
4 ways
Latency 2.2 ns
Shared LLC slice
8 MB
Latency 17.27 ns

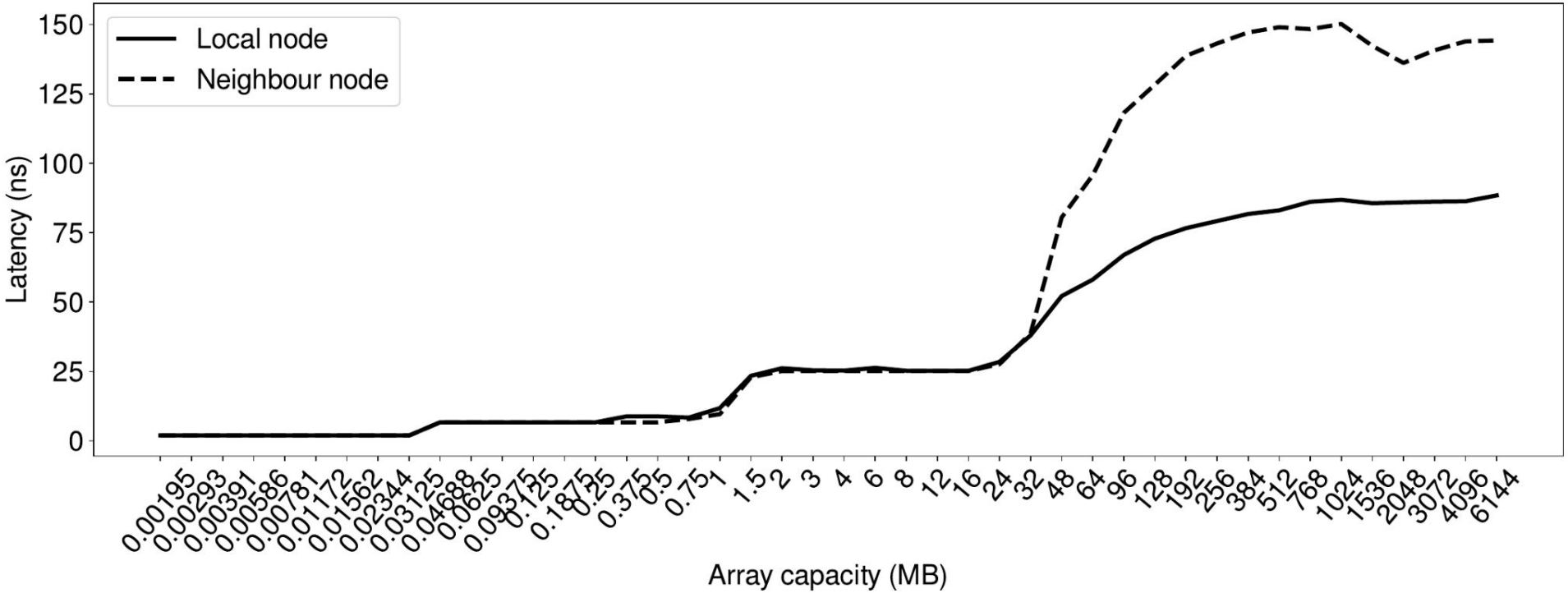
SKX


DDR4
48 GB
120 GB/s
Core
LLC
DDR4
48 GB
120 GB/s
Near mem.
latency
144 ns
Local mem.
latency
86.2 ns
Re-order buffer entries: 224
Instruction window entries: 97
Scalar registers: 180
Vector registers: 168
Private L1
32 KB
8 ways
Latency 1.91 ns
Private L2
1 MB
Latency 6.68 ns
Shared LLC slice
33 MB
Latency 25.14 ns

A64FX Latency

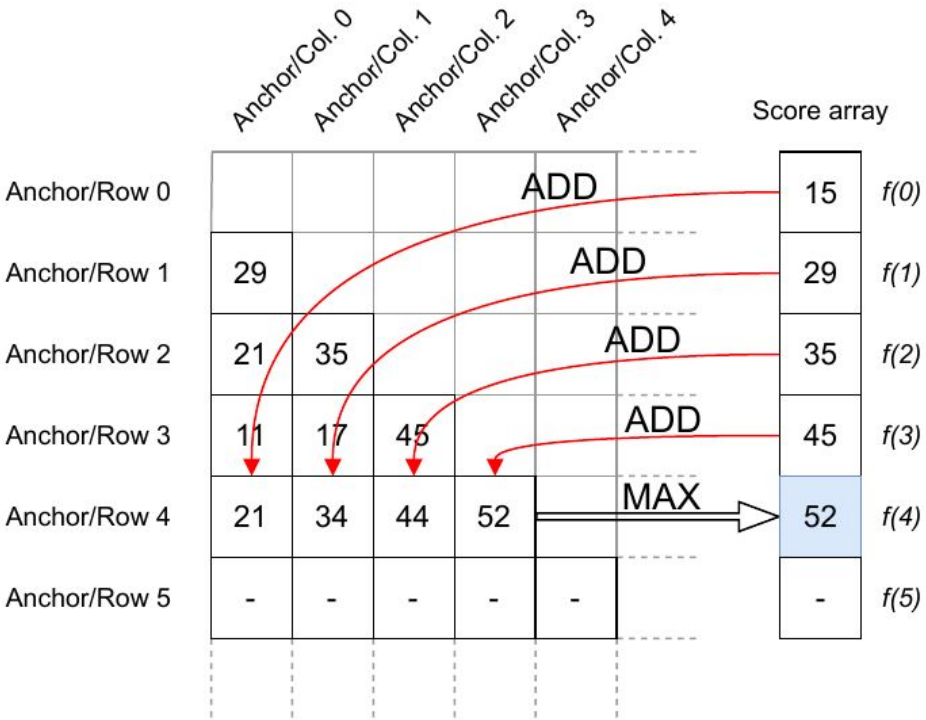


SKX Latency

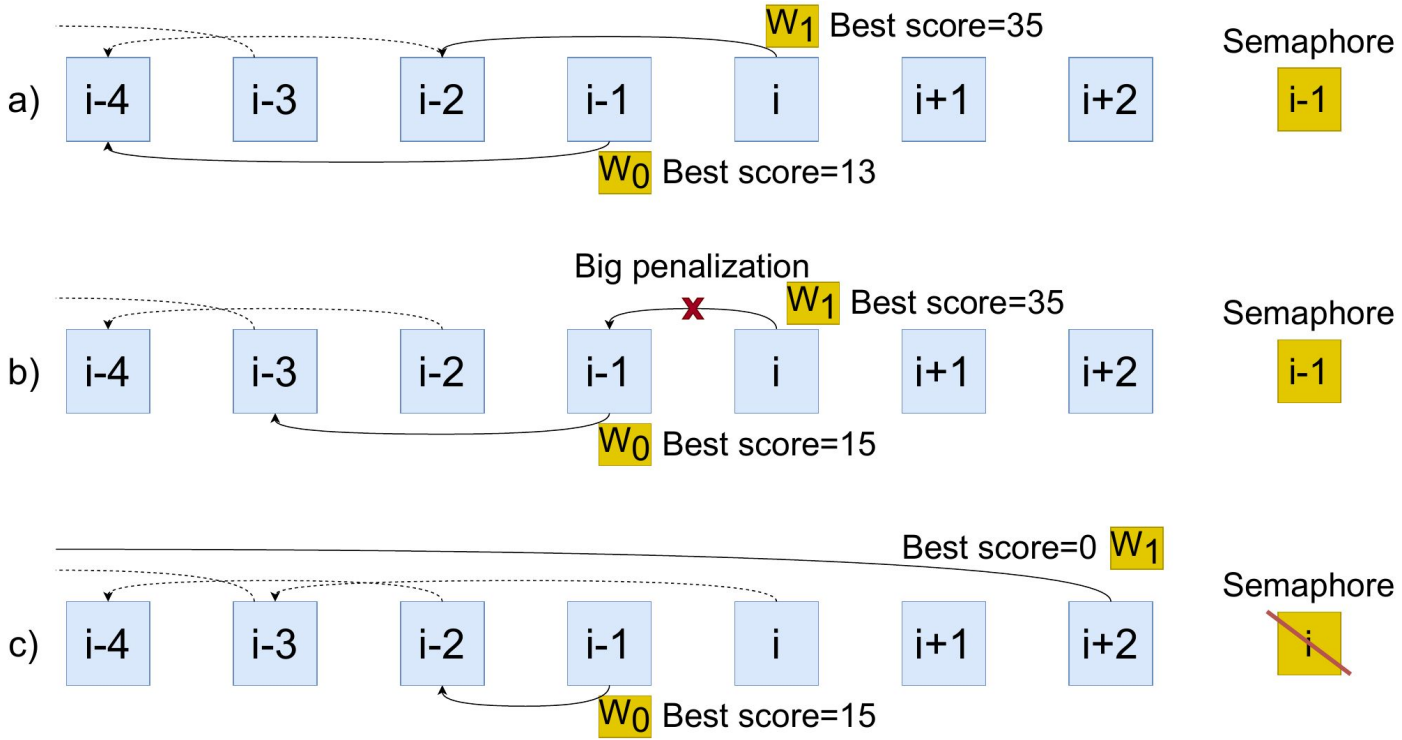

Local node
Neighbour node
Latency (ns)
150
125
100
75
50
25
0
0.00195
0.00293
0.00391
0.00586
0.00781
0.01172
0.01562
0.02344
0.03125
0.04688
0.0625
0.09375
0.125
0.1875
0.25
0.375
0.5
0.75
1
1.5
2
3
4
6
8
12
16
24
32
48
64
96
128
192
256
384
512
768
1024
1536
2048
3072
4096
6144
Array capacity (MB)

Squire

Squire Chain Dependencies



Squire Chain Global Counter



Squire API

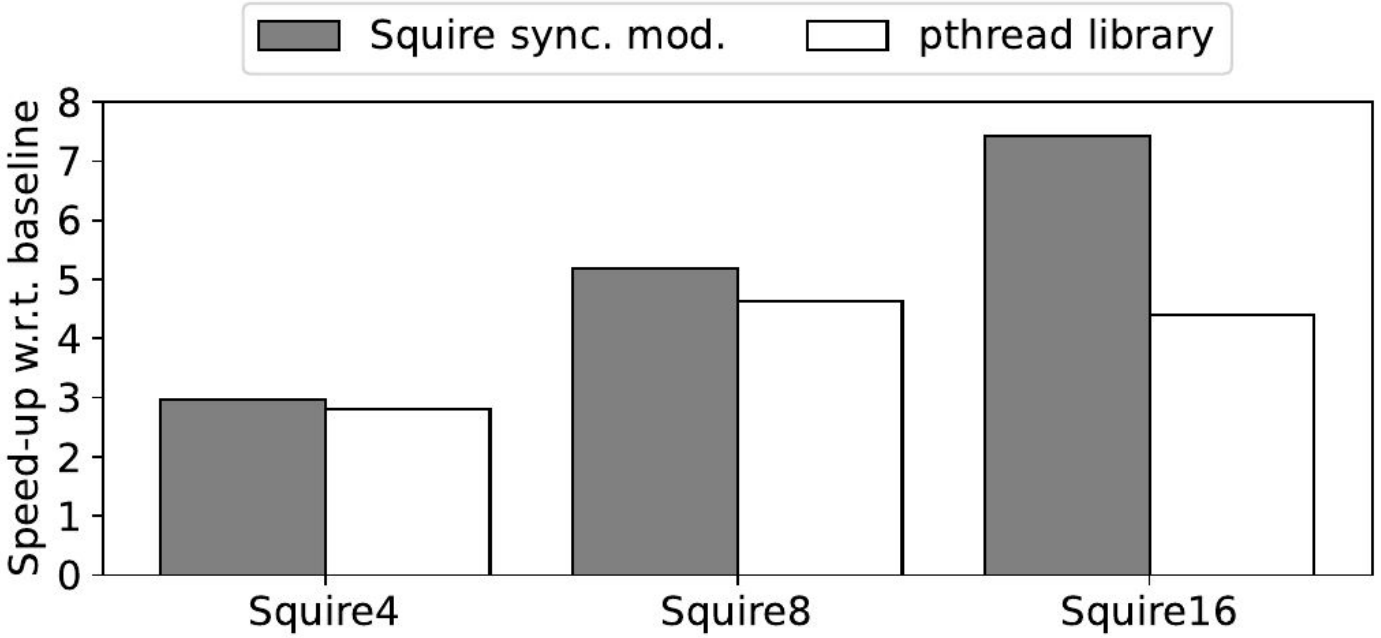
API call	Description	Caller
start_squire(f,a)	Squire executes *f* function with *a* arguments. *Counters* reset to 0.	Core
stop_worker()	Suspends the worker execution.	Workers
id_worker()	Returns the worker ID.	Workers
num_workers()	Returns the total number of workers.	Core / Workers
inc_lcounter(w)	Increments the *local counter w* by one.	Workers
inc_gcounter(w,s)	Increments the *global counter* by one.	Workers
wait_lcounter(w,s)	Waits until the *local counter w* is greater or equal to *s*.	Core / Workers
wait_gcounter(s)	Waits until the *global counter* is greater or equal to *s*.	Core / Workers

Squire Methodology

- We use Gem5 full system simulator to model Squire

Cores	8 Neoverse-N1-like Armv8 out-of-order cores 2.4 GHz, **each core is equipped with one Squire**
Structure entries	ROB: 224, LD/ST queues: 96/96, Inst. queue: 120
OoO Private L1 I&D	64 KB, 4-way, 1 cycle data access, 32 MSHRs
Private L2	512 KB, 8-way, 4 cycle data access, 64 MSHRs
Shared L3	Mostly exclusive, 8 slices of 1 MB, 16-way, 10 cycles data access, 128 MSHRs
Coherence protocol	MOESI-like AMBA 5 CHI specification
Network topology	4×4 2D mesh, 1 cycle routers, 1 cycle links
Memory	1 HBM2 stack, 300 GB/s
Squire	Each Squire consists of several workers, the synchronization module, the control registers, and the arbiter
Worker	Cortex-M35P-like Armv8 4-stage dual-issue in-order cores 2.4 GHz

Squire pthread Comparison



Squire Cache Size