# A closer look at the Fujitsu A64FX processor

Christie Alappat, Jan Laukemann, Thomas Gruber, Georg Hager, Gerhard Wellein, Nils Meyer, Tilo Wettig

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
University of Regensburg, Germany

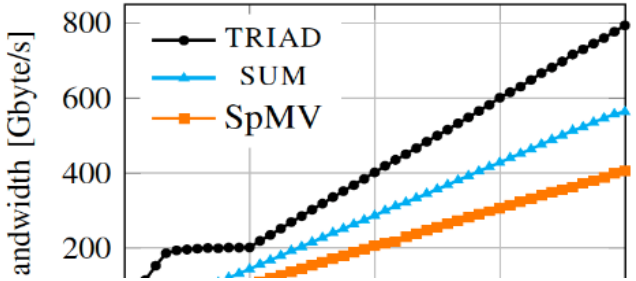
# PMBS 2020 Best Short Paper Award

## Performance Modeling of Streaming Kernels and Sparse Matrix-Vector Multiplication on A64FX

Christie Alappat, Jan Laukemann, Thomas Gruber,
Georg Hager, Gerhard Wellein
*Erlangen Regional Computing Center*
*Friedrich-Alexander-Universität Erlangen-Nürnberg*
Erlangen, Germany
christie.alappat@fau.de

Nils Meyer, Tilo Wettig
*Department of Physics*
*University of Regensburg*
Regensburg, Germany
tilo.wettig@ur.de

***Abstract*—The A64FX CPU powers the current #1 supercomputer on the Top500 list. Although it is a traditional cache-based multicore processor, its peak performance and memory bandwidth rival accelerator devices. Generating efficient code for such a new architecture requires a good understanding of its performance features. Using these features, we construct the Execution-Cache-Memory (ECM) performance model for the A64FX processor in the FX700 supercomputer and validate it**




2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), Atlanta, GA. DOI: 10.1109/PMBS51919.2020.00006, Preprint: arXiv:2009.13903

# Top500 list November 2020

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu<br>RIKEN Center for Computational Science<br>Japan | 7,630,848 | 442,010.0 | 537,212.0 | 29,899 |
| 2 | **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 2,414,592 | 148,600.0 | 200,794.9 | 10,096 |
| 3 | **Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox<br>DOE/NNSA/LLNL<br>United States | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | **Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC<br>National Supercomputing Center in Wuxi<br>China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | **Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia<br>NVIDIA Corporation<br>United States | 555,520 | 63,460.0 | 79,215.0 | 2,646 |

# Resource-based analytic modeling

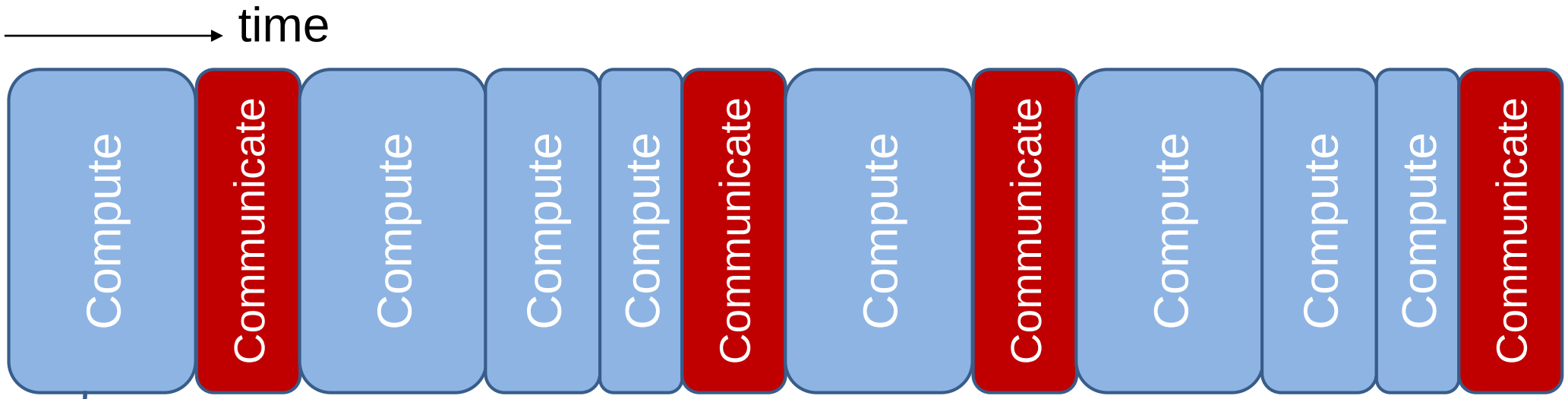
## Setting the stage

# How we approach perfomance

**How much of `$RESOURCE` does `$STUFF` need on `$HARDWARE`, and why?**

**→ Analytic, resource-based, first-principles models**

# Restrictions on **$STUFF**




```
parallel_for(i=0..N) { // N>>1
  update(data);
}
```

"Steady state"

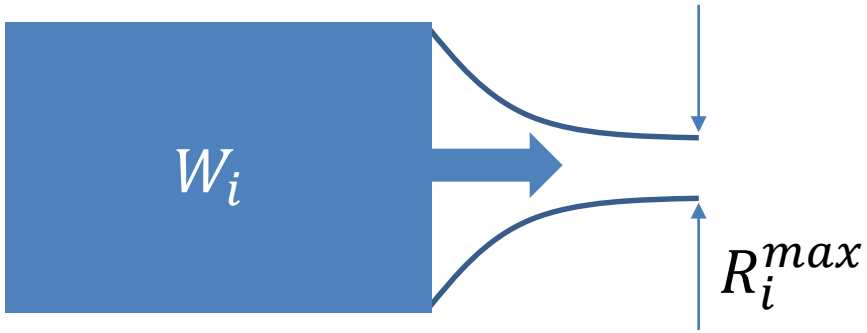
- Repetitive
- No startup/wind-down overhead

Runtime model: $T = f(\texttt{\$STUFF}, \texttt{\$HARDWARE})$

# A general view on resource bottlenecks

- What is the maximum performance when limited by a bottleneck?
- Resource bottleneck $i$ delivers resources at maximum rate $R_i^{max}$
- $W_i$ = needed amount of resources

- Minimum runtime: $T_i = \frac{W_i}{R_i^{max}} + \lambda_i$

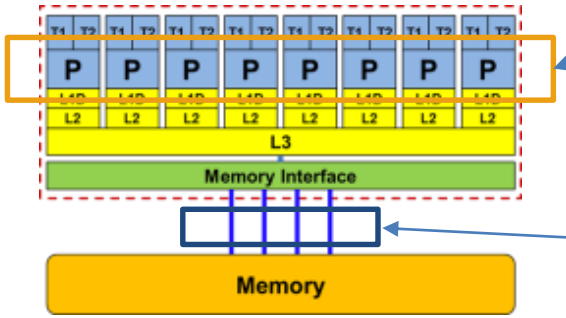



- Multiple bottlenecks → multiple min. runtimes: $T_{\text{expect}} = f(T_1, \dots T_n)$
- Overall performance: $P_{\max} = \frac{W}{T_{\text{expect}}}$

# A bottleneck model of computing

## Example: two bottlenecks

```
#pragma omp parallel for
for(i=0; i<10^7; ++i)
  a[i] = a[i] + s * c[i];
```




8-core CPU
(3 GHz Intel Sandy Bridge)

$$R_{flops}^{max} = 192\,\frac{\text{Gflops}}{\text{s}}$$

$$R_{BW}^{max} = 40\,\frac{\text{Gbyte}}{\text{s}}$$

$$W_{flops} = 2 \times 10^7 \text{ flops}$$

$$W_{BW} = 3 \times 8 \times 10^7 \text{ bytes}$$

$$T_{flops} = \frac{2 \times 10^7 \text{ flops}}{192\,\frac{\text{Gflops}}{\text{s}}} = 104\ \mu s$$

$$T_{BW} = \frac{2.4 \times 10^8 \text{ bytes}}{40\,\frac{\text{Gbyte}}{\text{s}}} = 6.0 \text{ ms}$$

# Bottleneck models

How do we reconcile the multiple bottlenecks?
I.e., what is the functional form of $f(T_1, \dots T_n)$?

→ pessimistic model (no overlap): $f(T_1, \dots T_n) = \sum_i T_i$
→ optimistic model (full overlap): $f(T_1, \dots T_n) = \max(T_1, \dots T_n)$

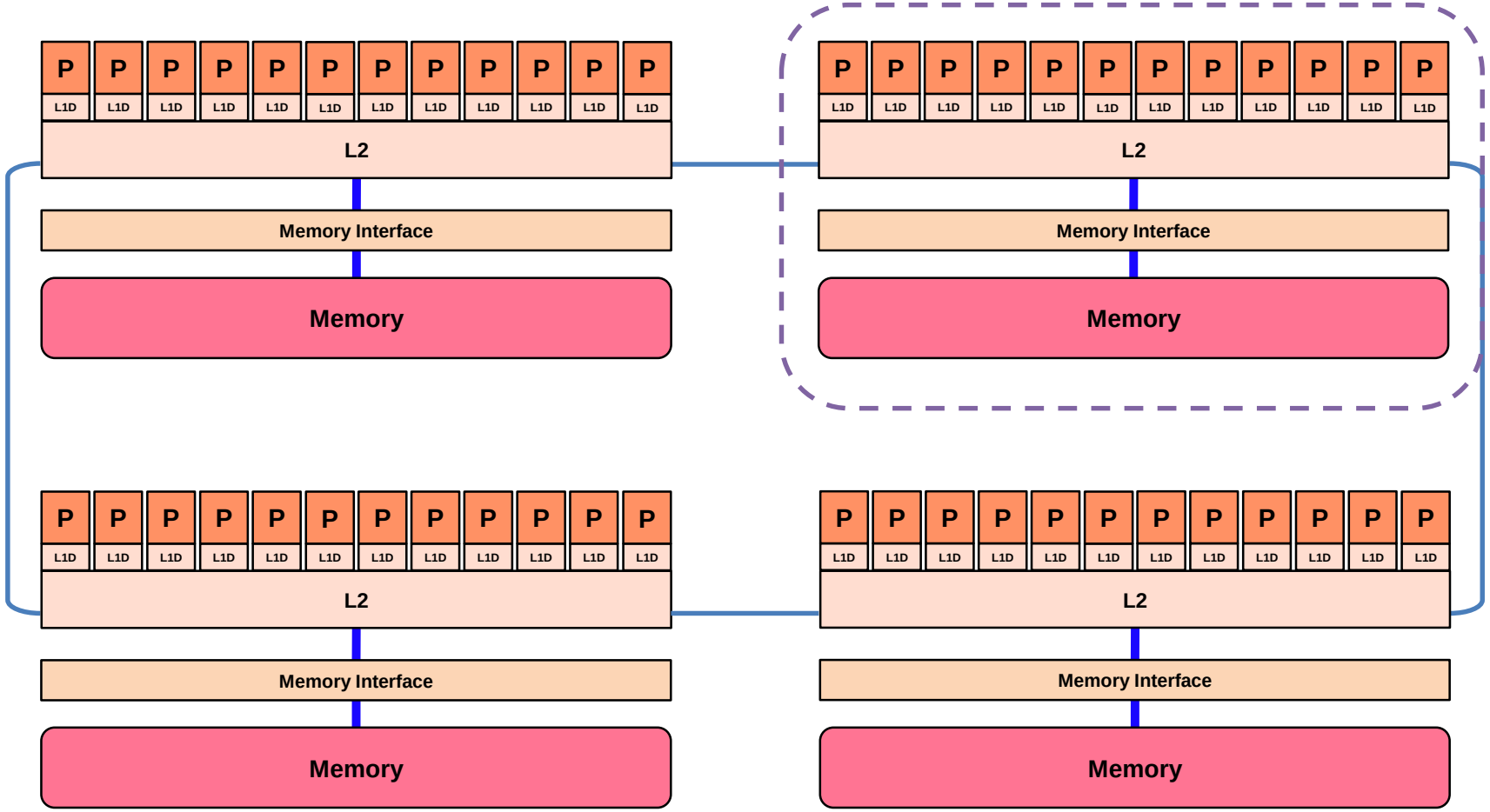
Roofline model (Williams et al., 2008)

Roofline for our example: $T_{\min} = \max\left(T_{flops}, T_{BW}\right) = 6\ \mathrm{ms}$

Maximum performance ("light speed"): $P_{\max} = \frac{2\times10^7}{6.0\times10^{-3}} \frac{\mathrm{flops}}{\mathrm{s}} = 3.3\ \mathrm{Gflop/s}$

# A64FX architecture

# A64FX – FX700

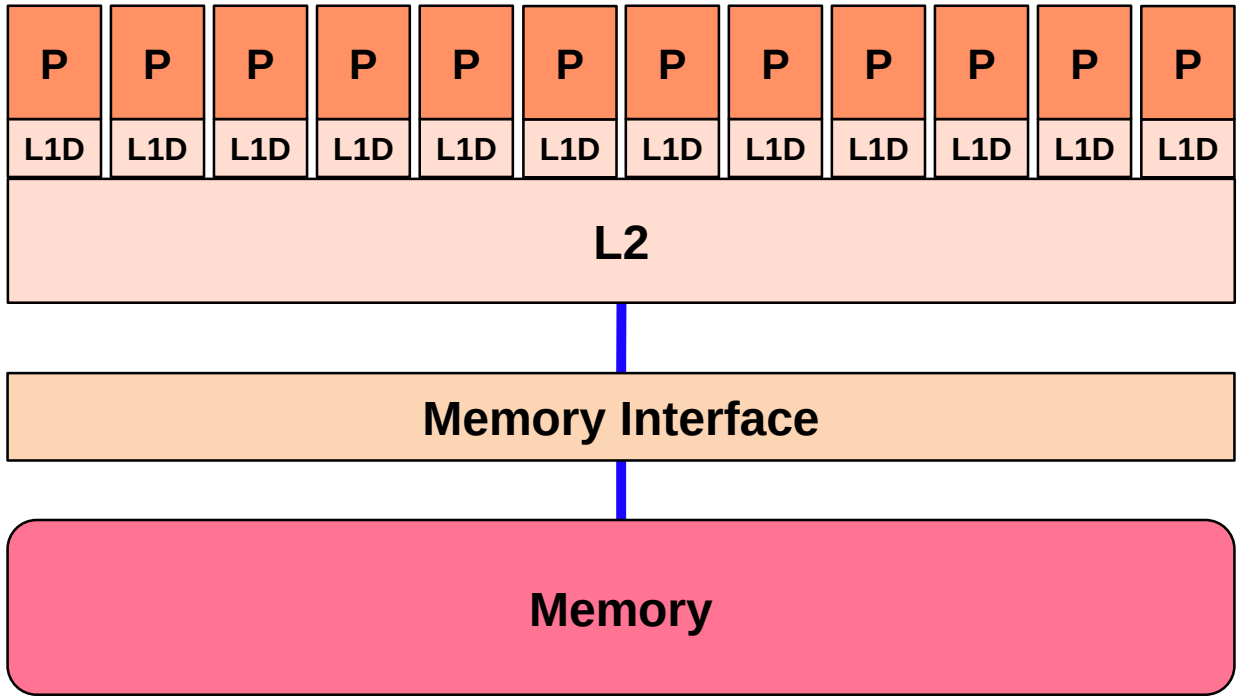



1 CMG (Core-Memory Group)

4 CMGs per node

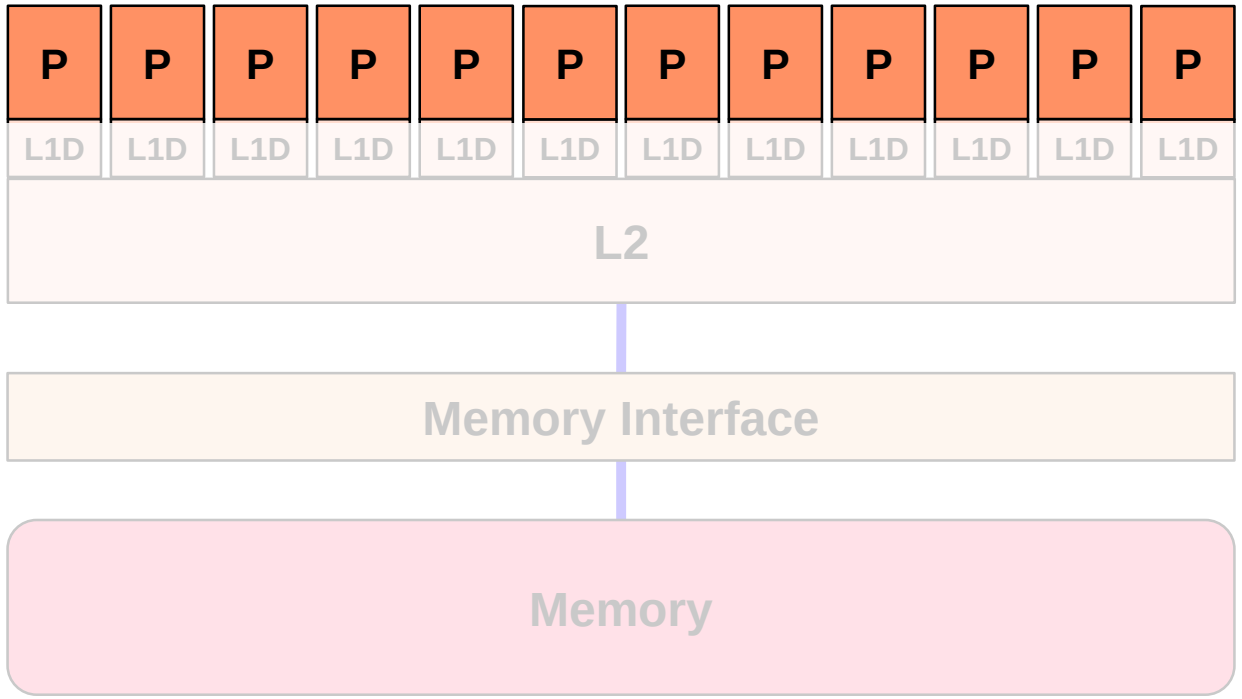
48 cores per node

# A64FX – FX700




1 CMG

# A64FX – FX700




1 CMG

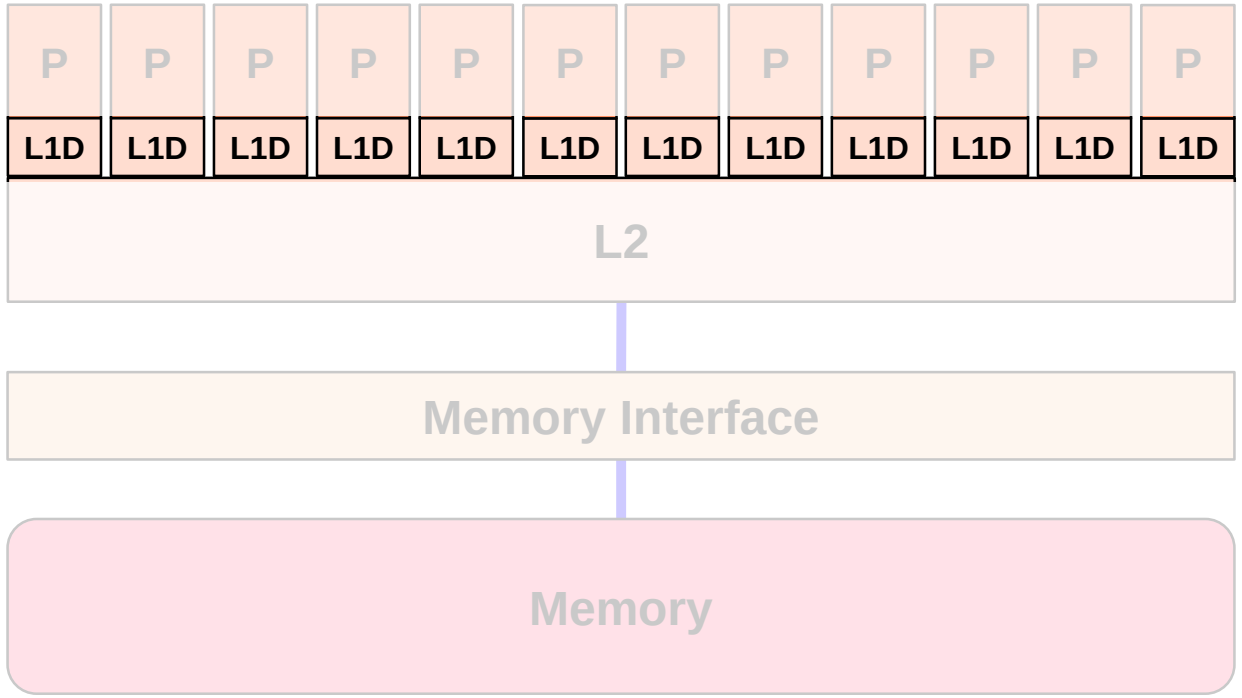
## Core

- Clock : 1.8 GHz
- Instruction set : Armv8.2-A+SVE
- Maximum VL : 512 bit (8 double)

GCC v10.1.1 flags:

```
-msve-vector-bits=512 -march=armv8.2-a+sve
```

# A64FX – FX700

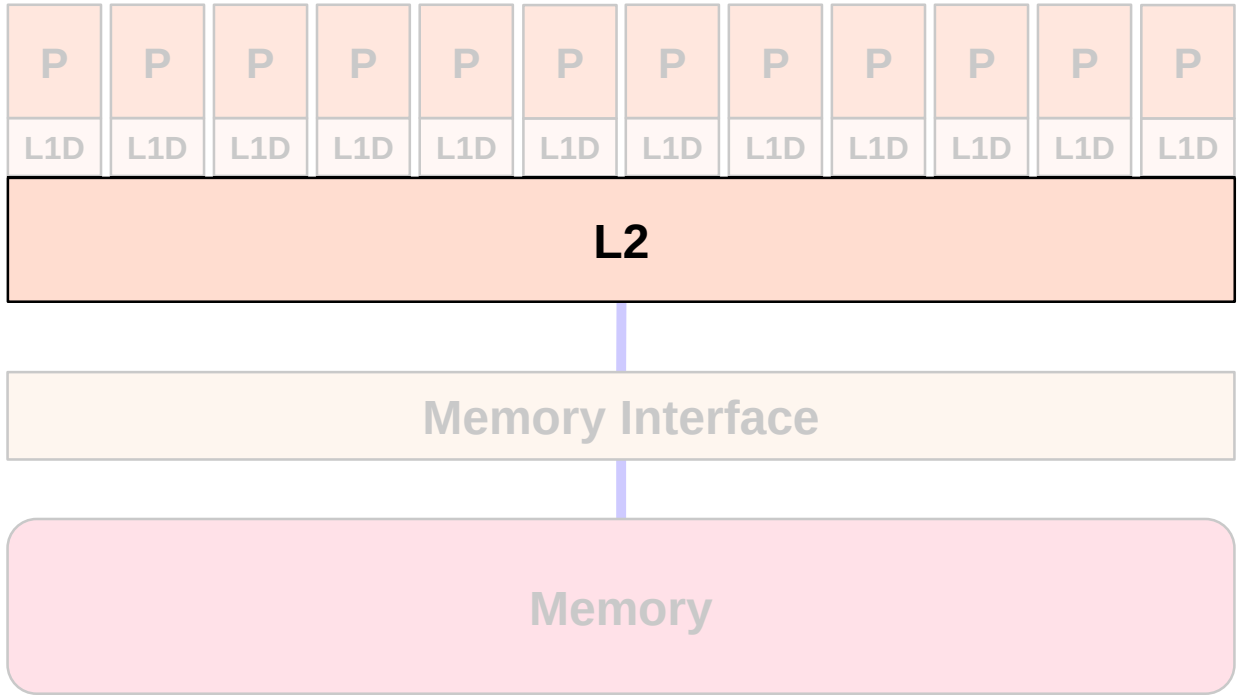



1 CMG

## L1D cache

- Size : 64 KiB
- Topology : Private cache
- Cache line size : 256 bytes

# A64FX – FX700

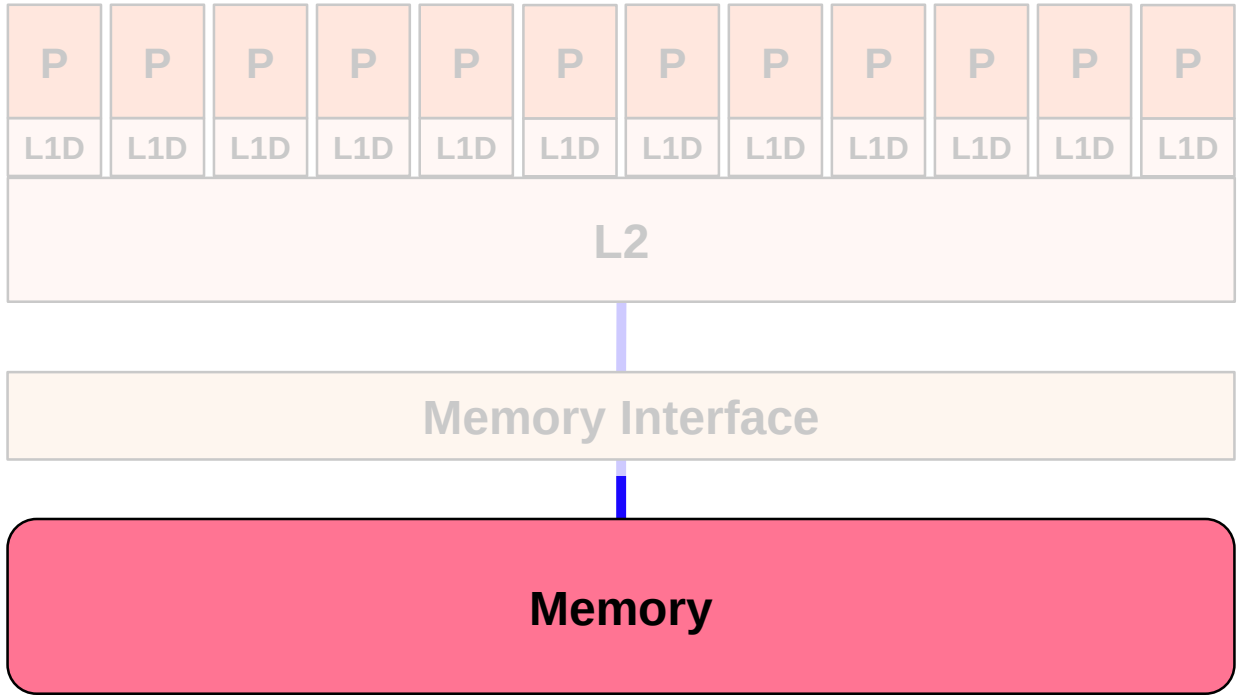



1 CMG

## L2 cache

- Size : 8 MiB
- Topology : Shared within 1 CMG
- Cache line size : 256 bytes

# A64FX – FX700

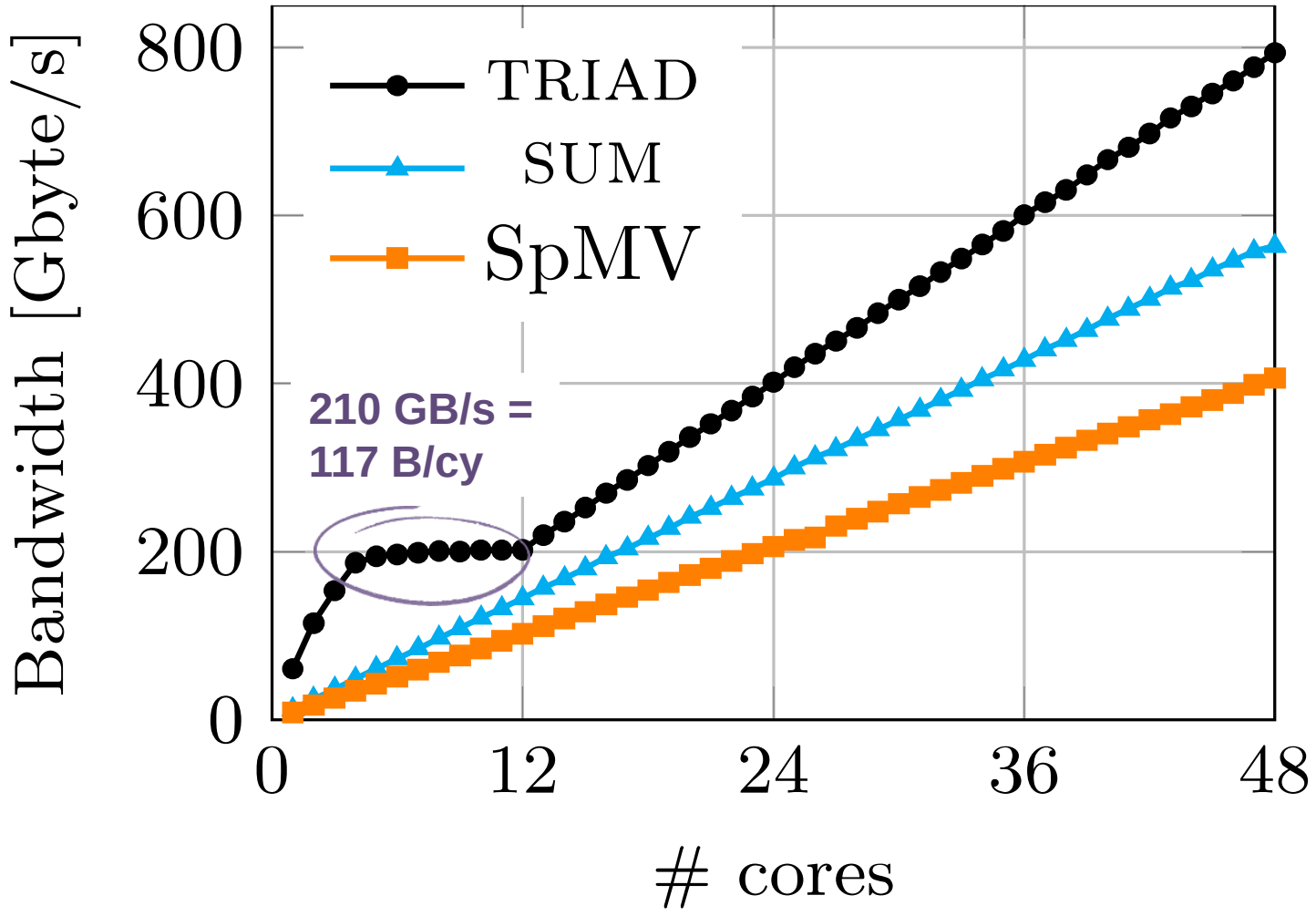



1 CMG

## Main Memory

- Size : 4 x 8 GiB
- Type : HBM2

# Motivation

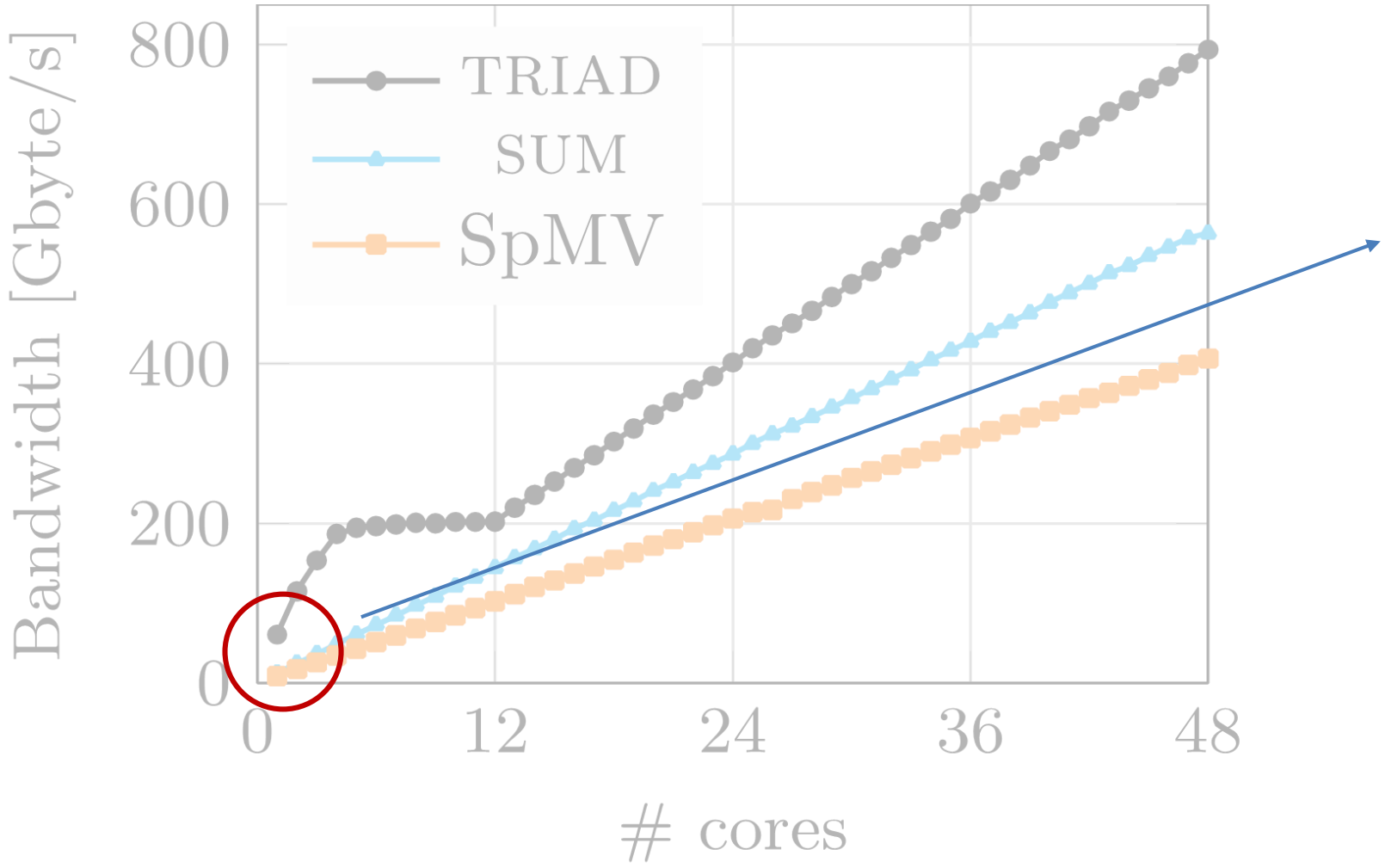


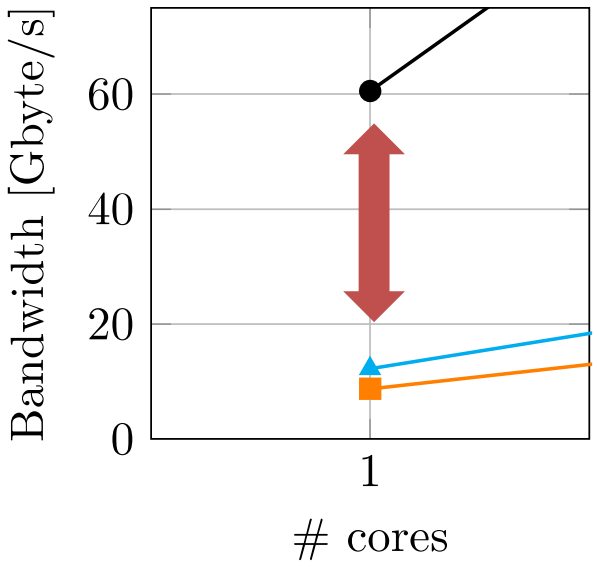

Clear memory bandwidth saturation for STREAM TRIAD (`a[i] = b[i] + s*c[i]`)

But why not for SUM (`s += a[i]`) and SpMV (`b[:] = A[:,:]x[:]`) ?

Thread pinning : Compact

# Motivation



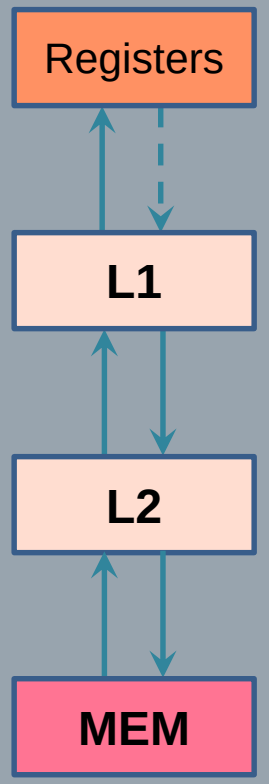




Understanding single-core performance is the key!

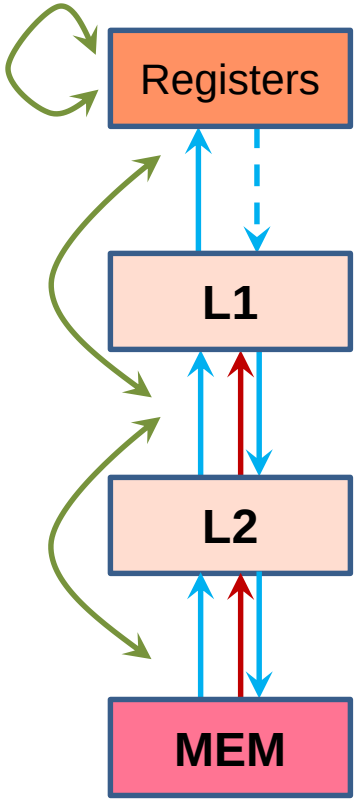
Thread pinning : Compact

# Single-core analysis

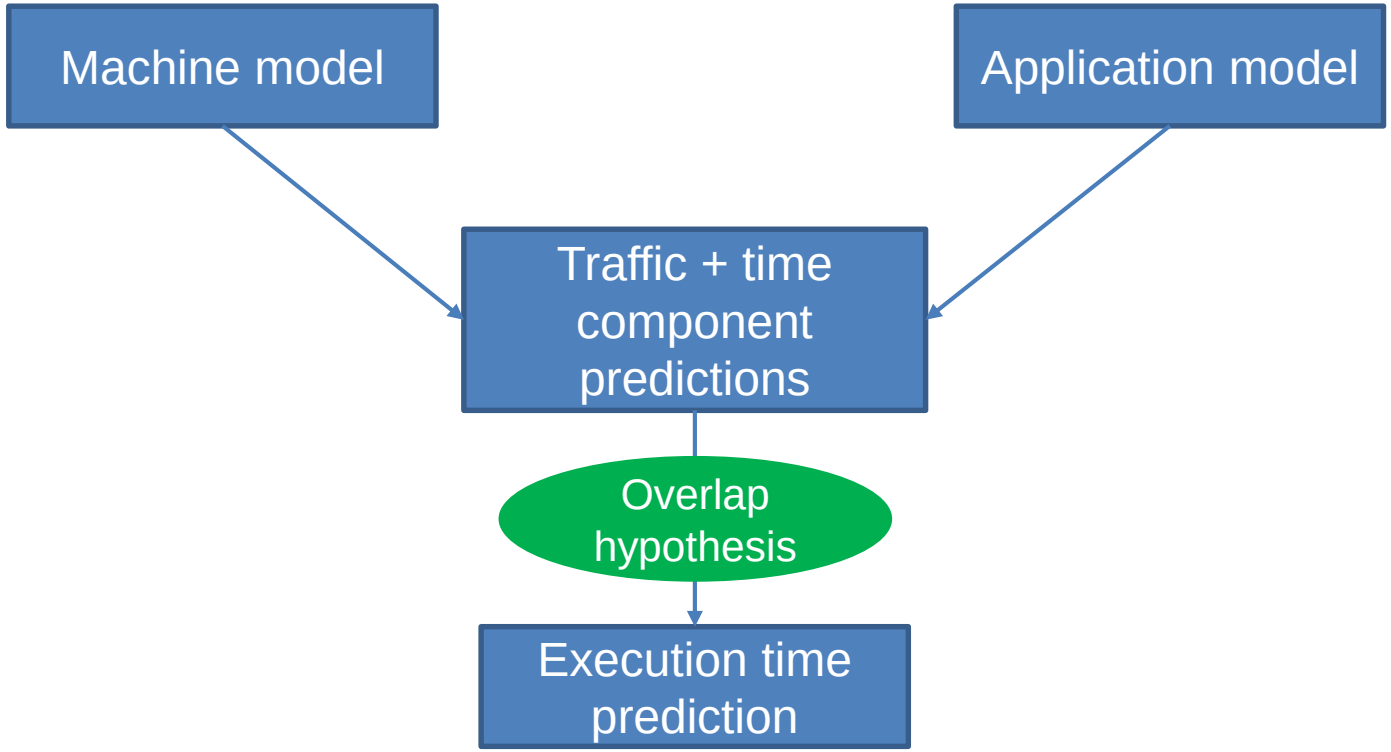
## ECM model



# Single core ECM model

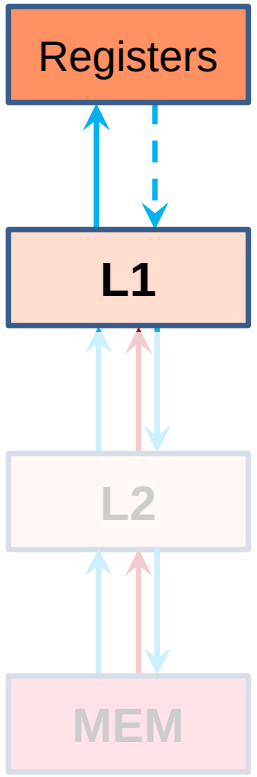



Execution-Cache-Memory (ECM) model helps us to understand and analyze the single-core performance.

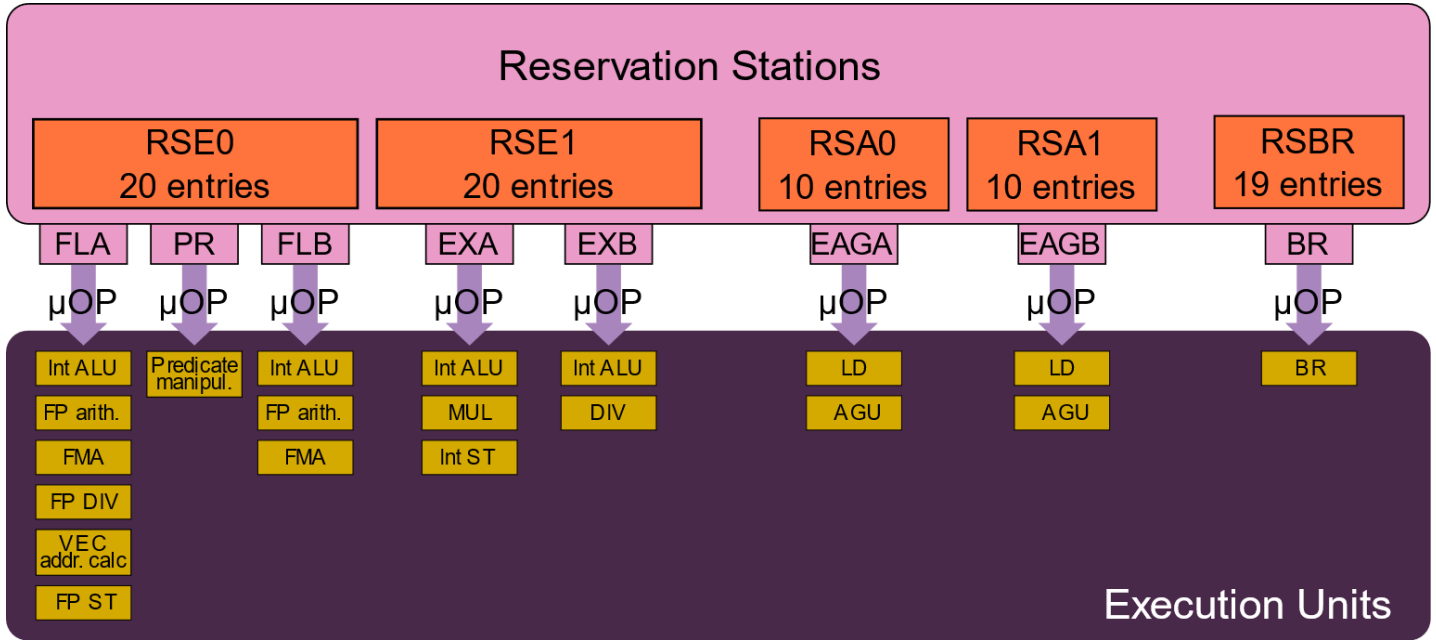



Hofmann et.al.: *Bridging The Architecture Gap: Abstracting Performance-Relevant Properties Of Modern Server Processors*, https://doi.org/10.14529/jsfi200204

# In-core prediction




## Machine knowledge



# In-core prediction

## Application knowledge

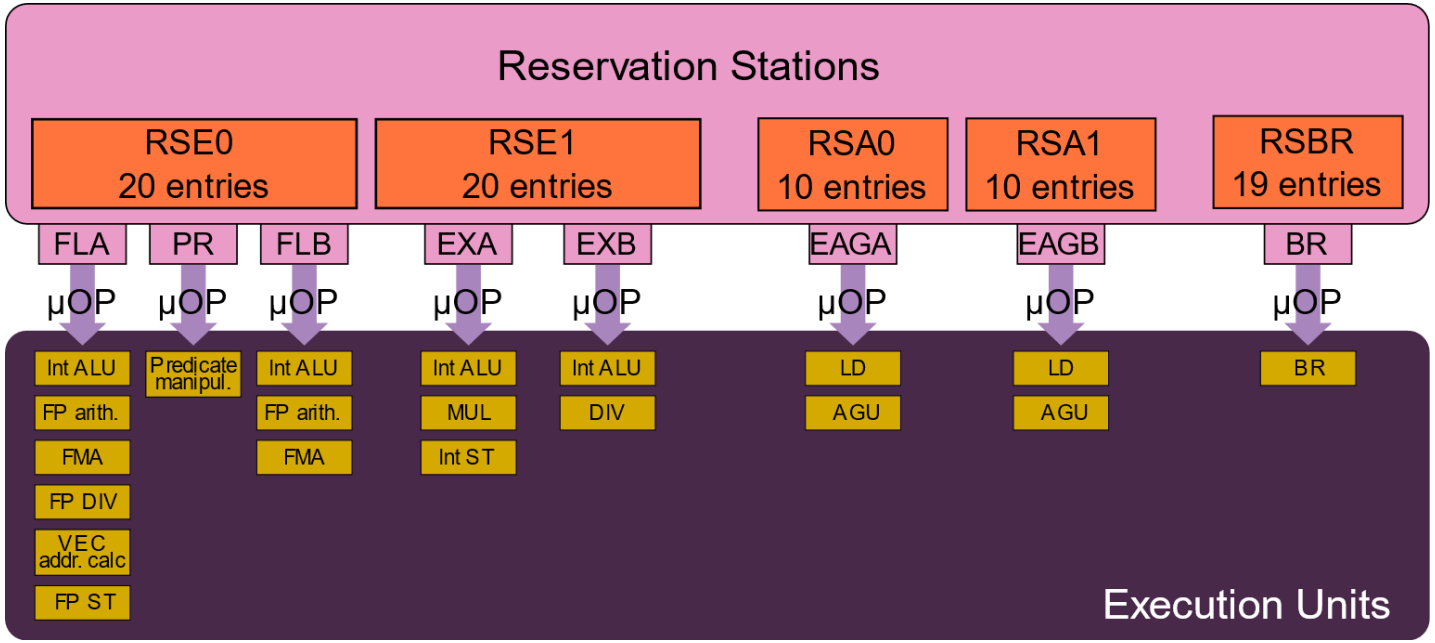
STREAM TRIAD

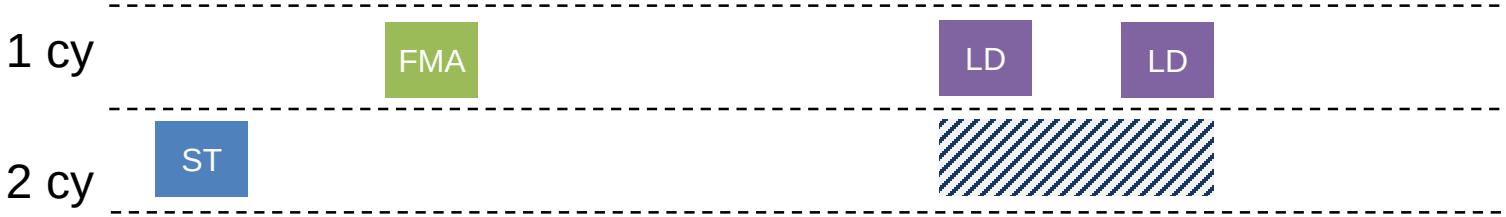
```
a[i] = b[i] + s * c[i]
```

```
.L18:
  ld1d z4.d, p5/z, [x21, x9, lsl 3]
  ld1d z5.d, p5/z, [x20, x9, lsl 3]
  fmad z5.d, p5/m, z2.d, z4.d
  st1d z5.d, p5, [x19, x9, lsl 3]
  add x8, x9, 8
  whilelo p5.d, w8, w7
  b.any .L18
```

2cy / VL

## Machine knowledge



# In-core prediction

## Application knowledge

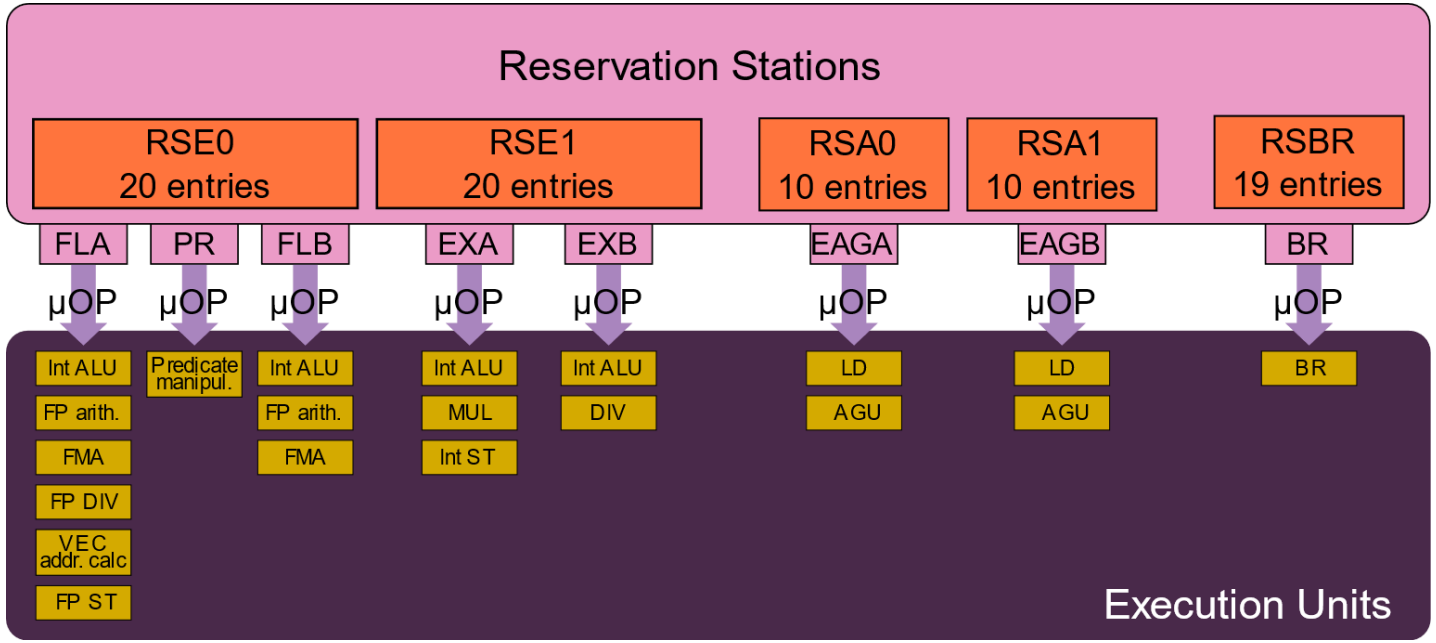
STREAM TRIAD

```
a[i] = b[i] + s * c[i]
```

```
.L18:
  ld1d z4.d, p5/z, [x21, x9, lsl 3]
  ld1d z5.d, p5/z, [x20, x9, lsl 3]
  fmad z5.d, p5/m, z2.d, z4.d
  st1d z5.d, p5, [x19, x9, lsl 3]
  add x8, x9, 8
  whilelo p5.d, w8, w7
  b.any .L18
```

2cy / VL

## Machine knowledge



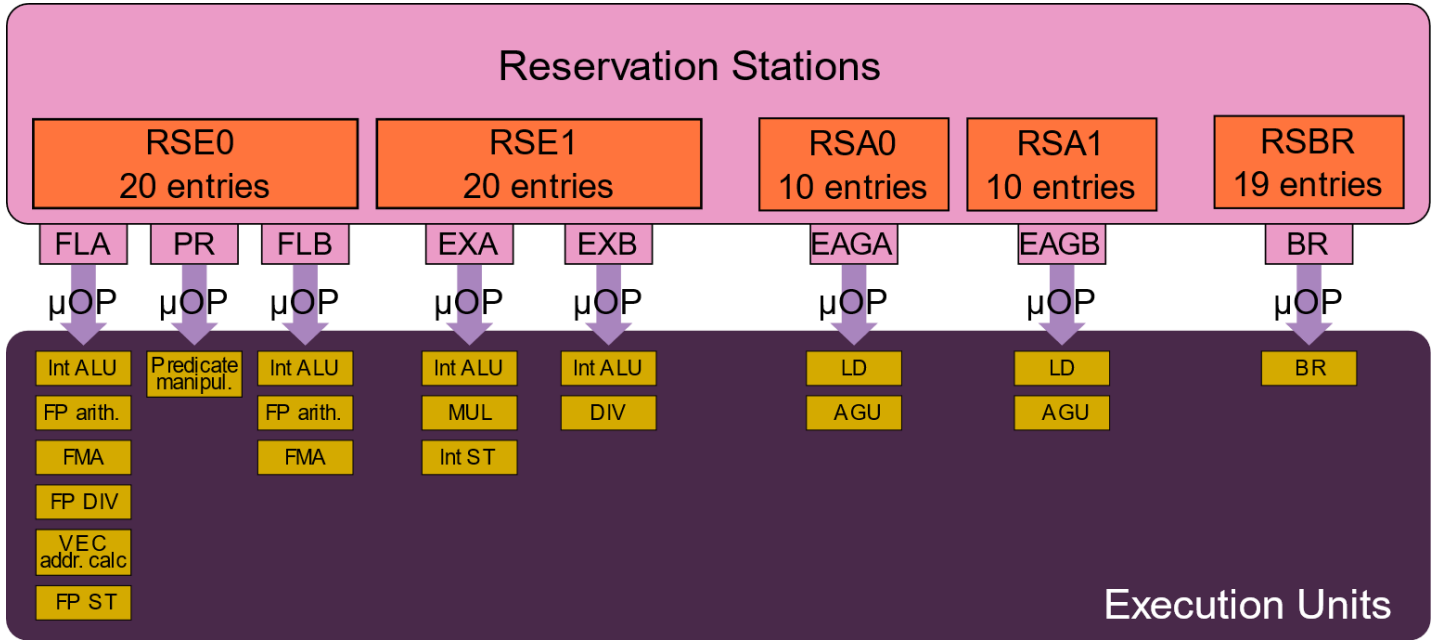

OSACA

Static analysis and prediction of in-core contribution

https://github.com/RRZE-HPC/OSACA

# In-core prediction

| Instruction | Reciprocal Throughput [cy] | Latency [cy] |
|---|---|---|
| `ld1d` | 0.5 | 11 |
| `st1d` | 1.0 | – |
| `fadd` | 0.5 | 9 |
| `fmad` | 0.5 | 9 |
| `faddv` | 11.5 | 49 |




OSACA

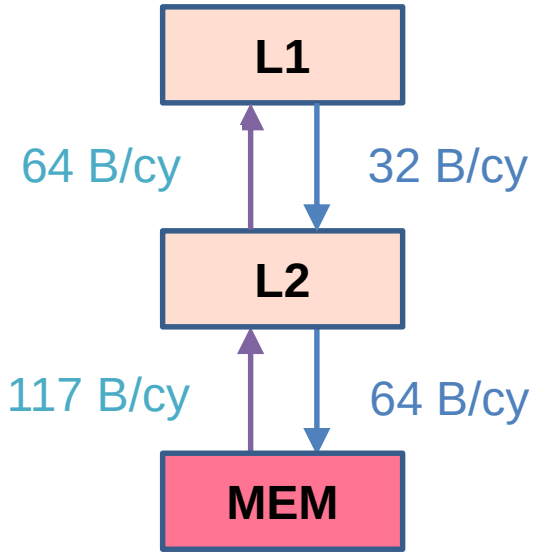
Static analysis and prediction of in-core contribution

https://github.com/RRZE-HPC/OSACA

# Data transfer

Machine knowledge

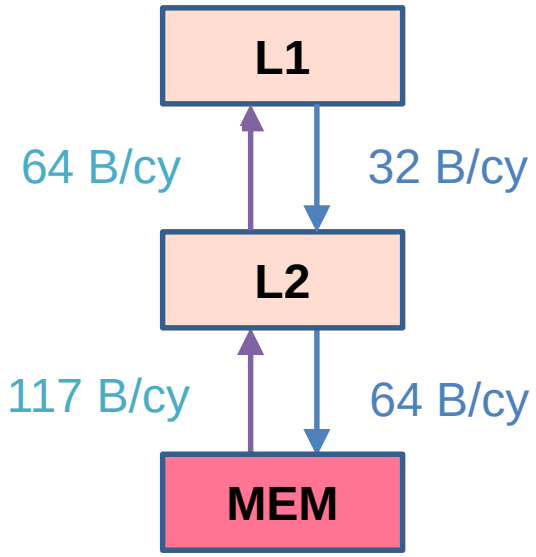
FX700



# Data transfer

Machine knowledge

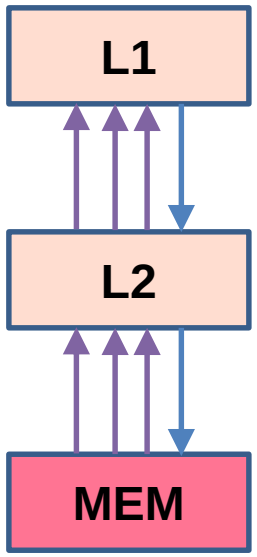
FX700




Application knowledge

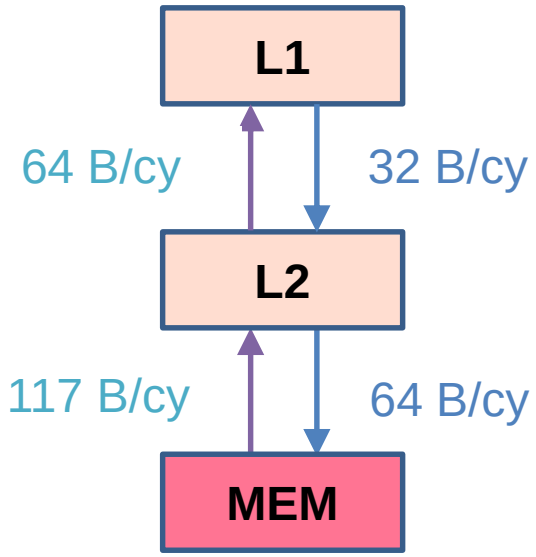
STREAM TRIAD

```
a[i] = b[i] + s*c[i]
```



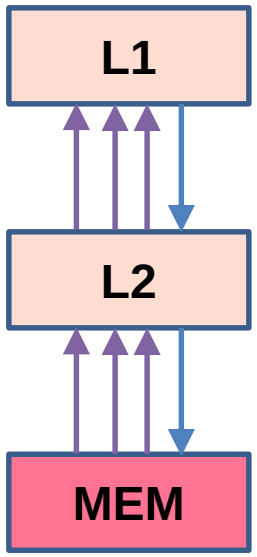
# Data transfer

## Machine knowledge
FX700

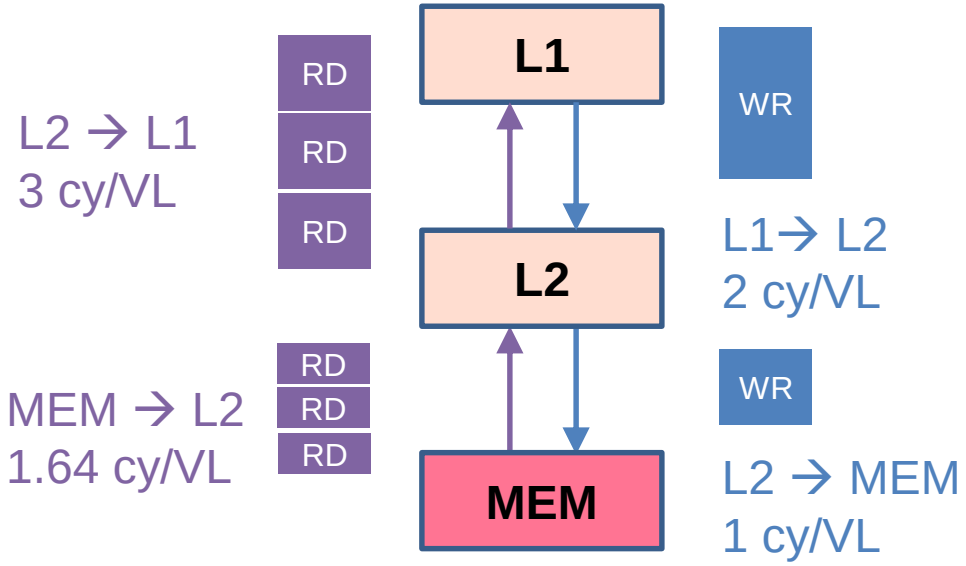



## Application knowledge
STREAM TRIAD

```
a[i] = b[i] + s*c[i]
```

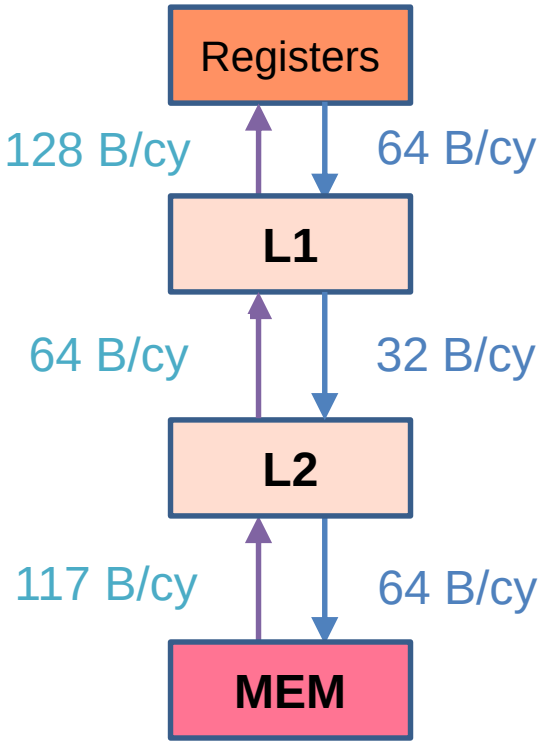



## ECM prediction
STREAM TRIAD on FX700



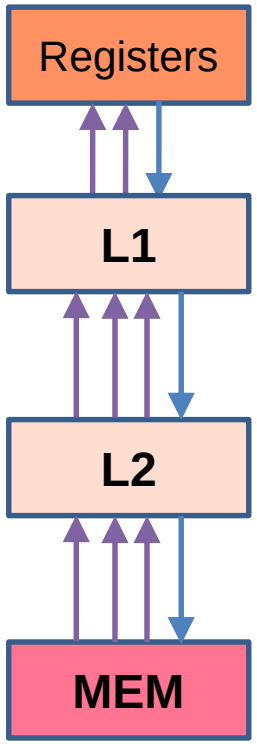
# ECM model

## Machine knowledge
FX700

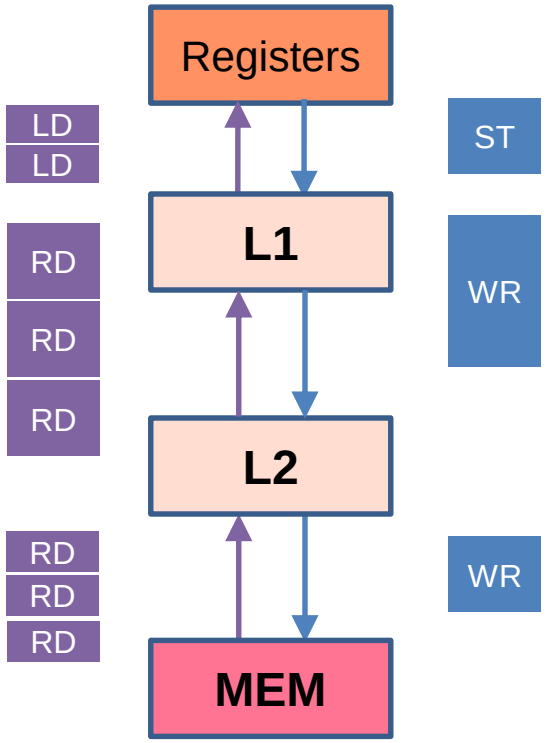



## Application knowledge
STREAM TRIAD

```
a[i] = b[i] + s*c[i]
```




## ECM prediction
STREAM TRIAD on FX700

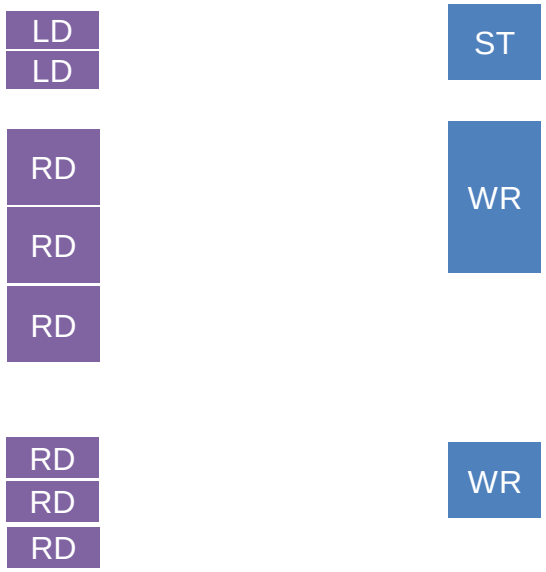


# ECM model

How do these boxes overlap?

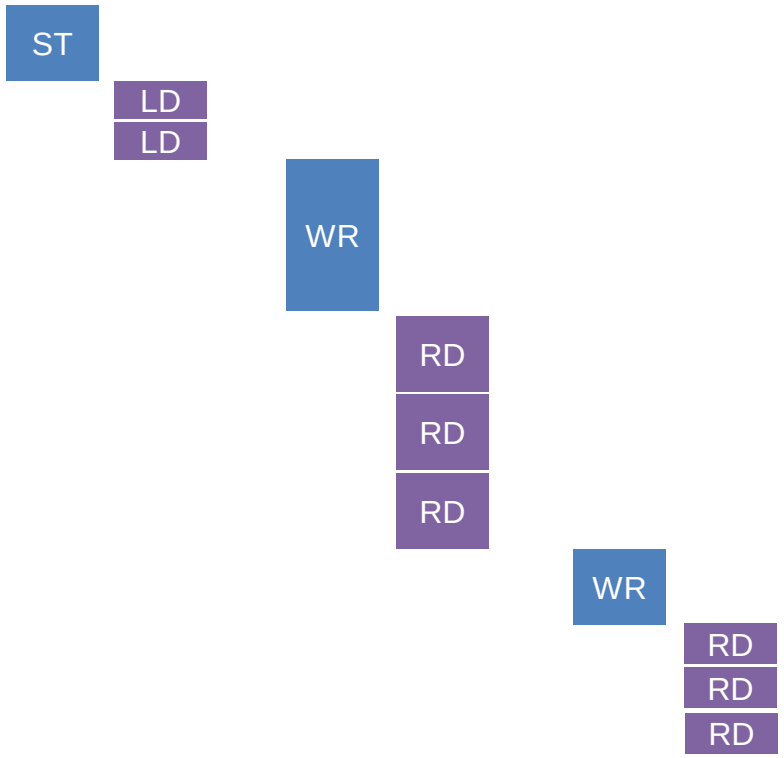
ECM prediction

STREAM TRIAD on FX700



# ECM model

## Hypothesis 1 : No overlap

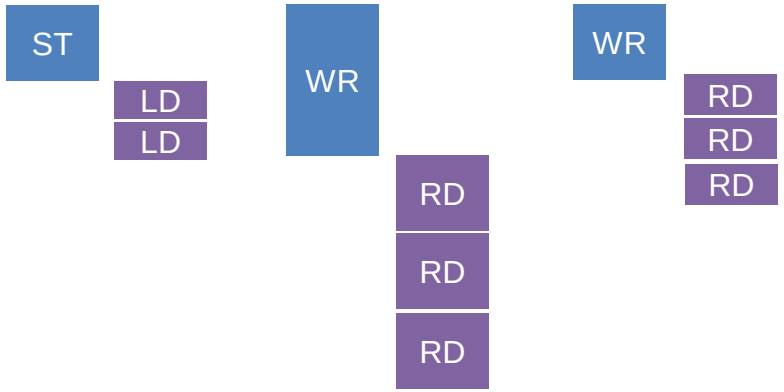


# ECM model

## Hypothesis 2 : Full overlap

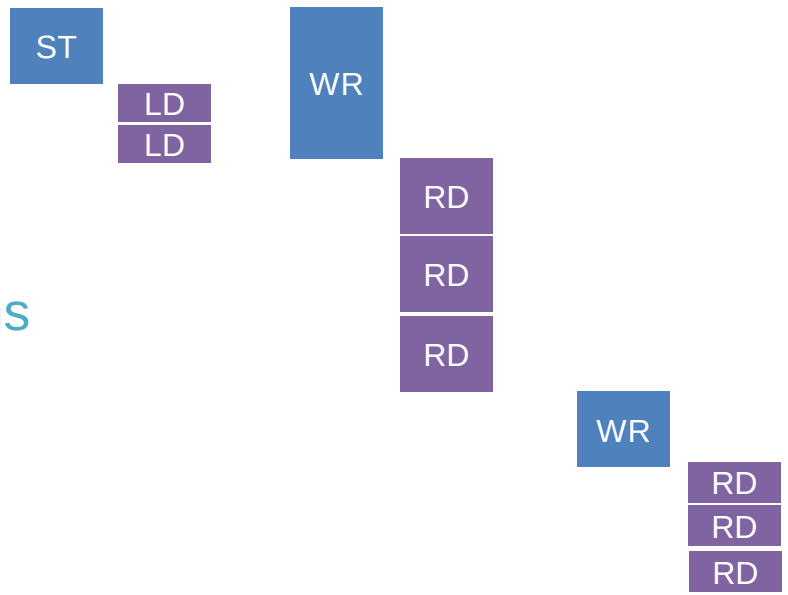
# ECM model

Hypothesis 3 : Full overlap + half-duplex



# ECM model

Hypothesis 4 : L1L2 overlap + half-duplex



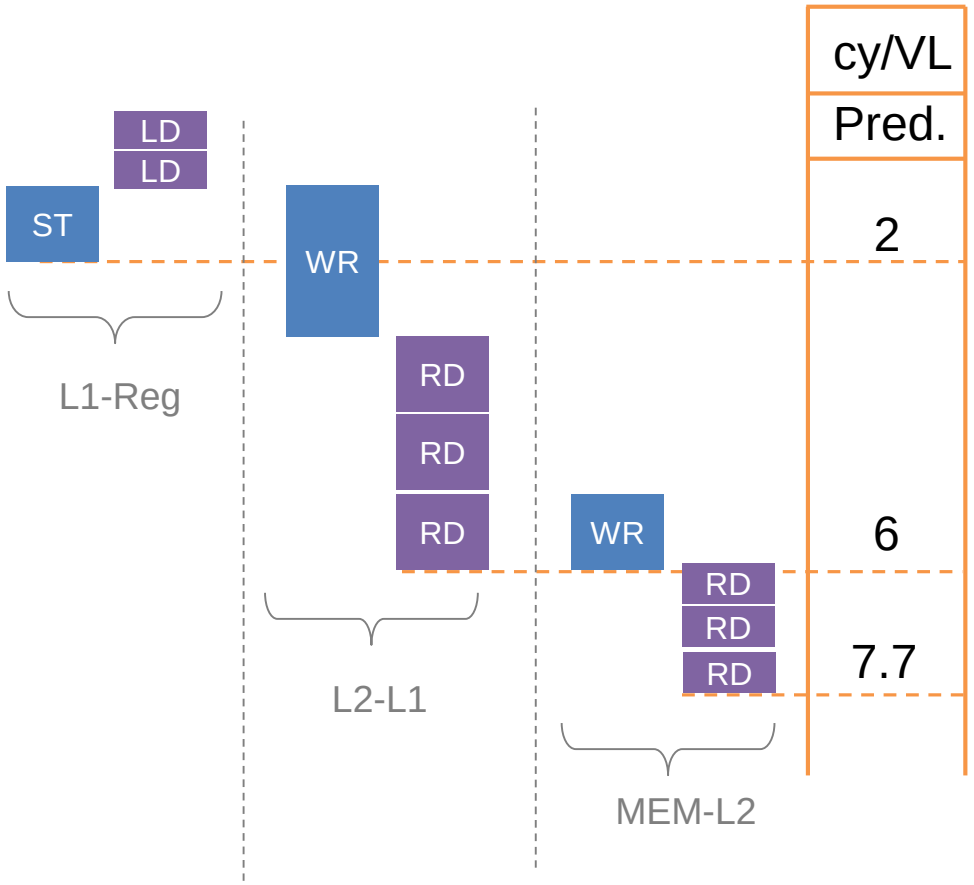

There are numerous combinations

How do we find the correct one?

→ Compare measurements with predictions

# ECM model

The best hypothesis for FX700

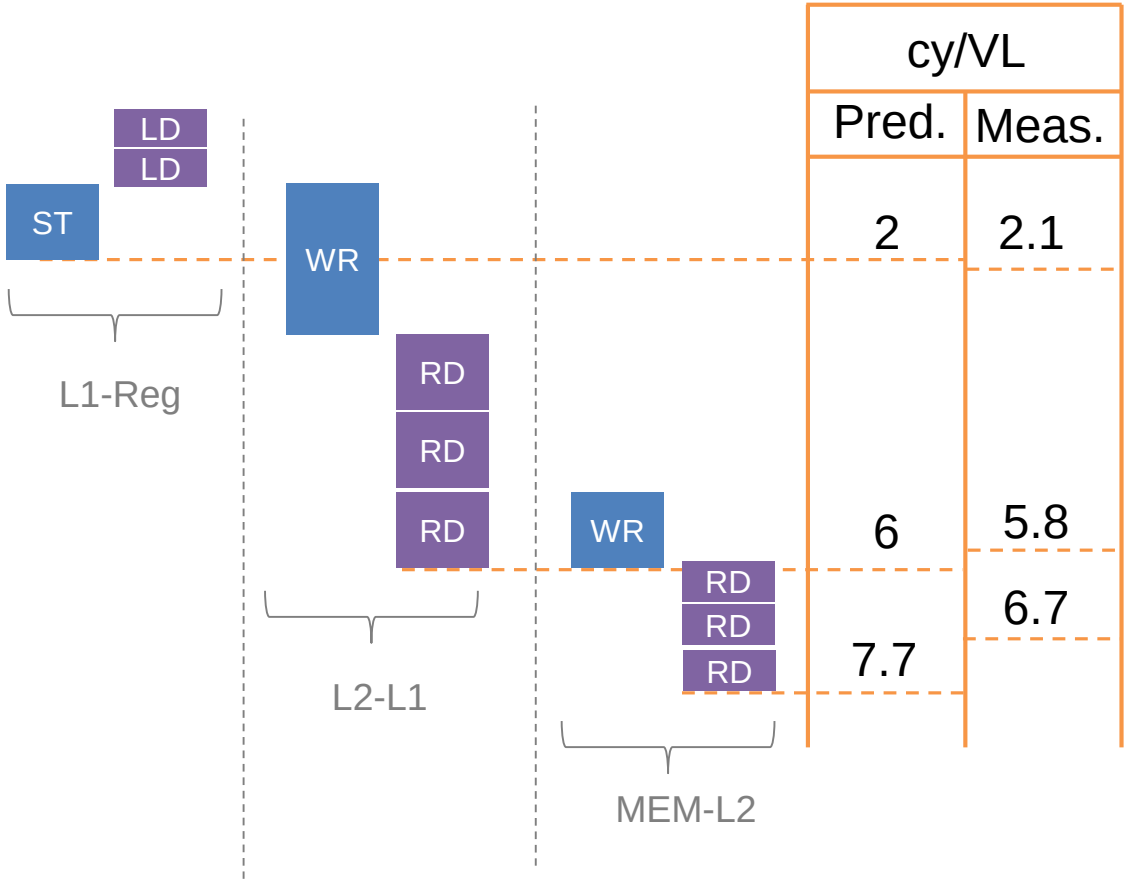



How do we find the correct one?

→ Compare measurements with predictions

# ECM model

The best hypothesis for FX700

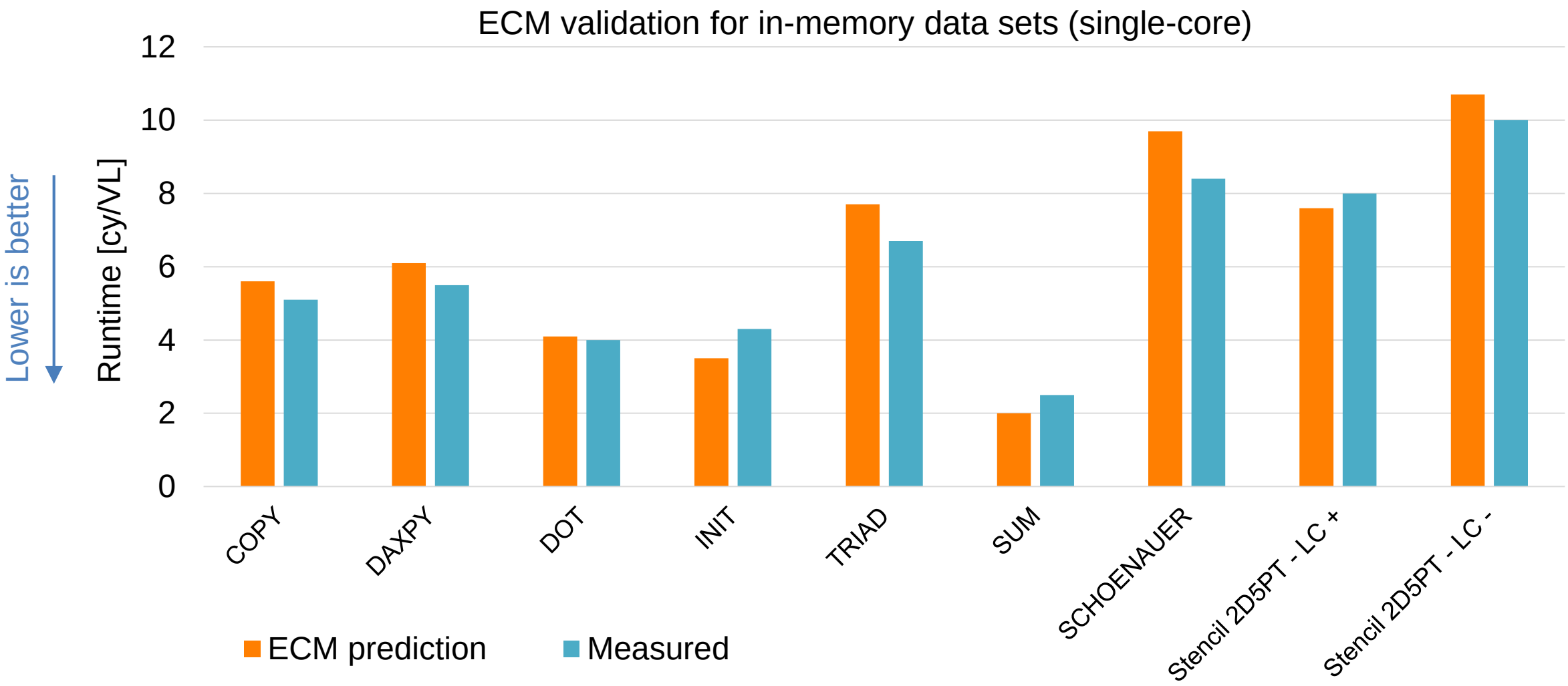



How do we find the correct one?

→ Compare measurements with predictions

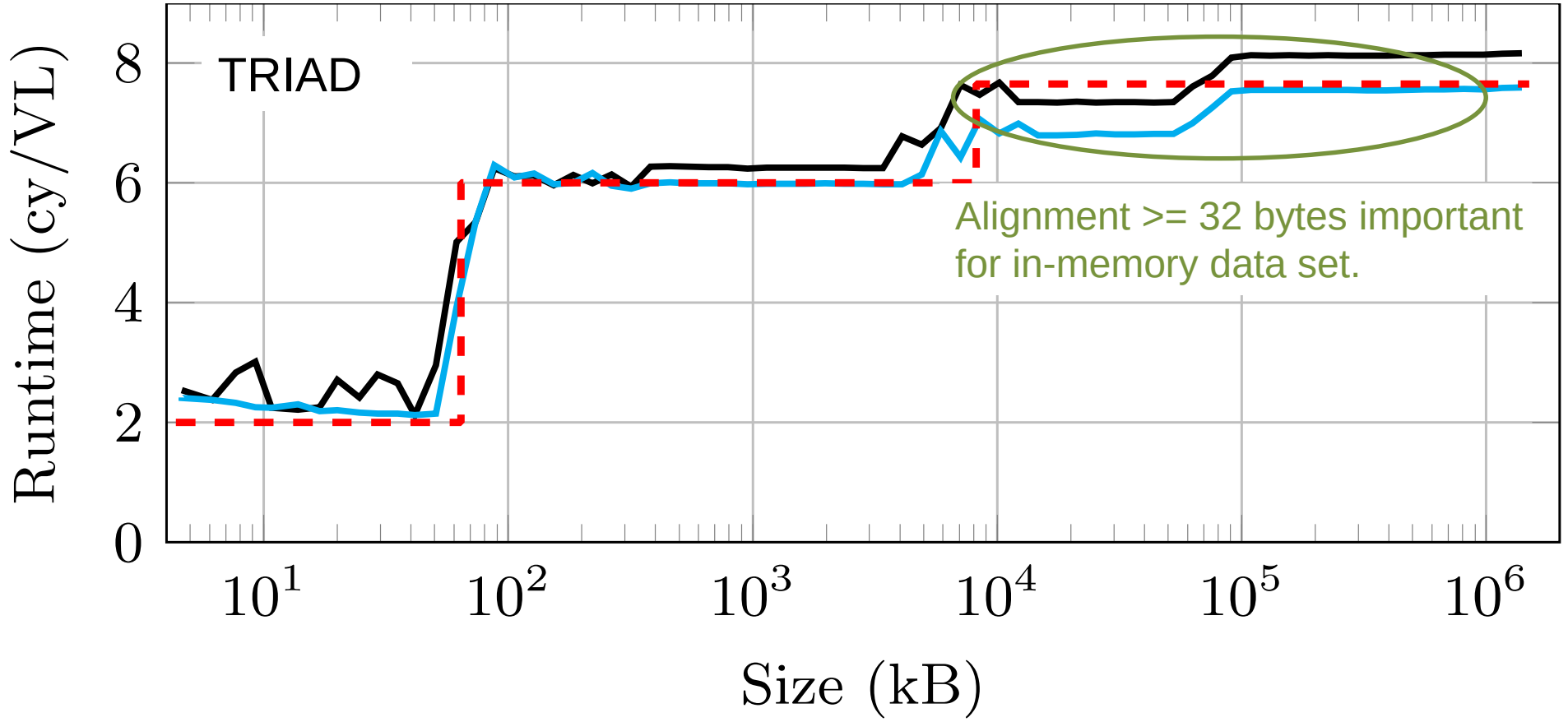
Hofmann et.al.: *Bridging The Architecture Gap: Abstracting Performance-Relevant Properties Of Modern Server Processors*, https://doi.org/10.14529/jsfi200204

# Model validation



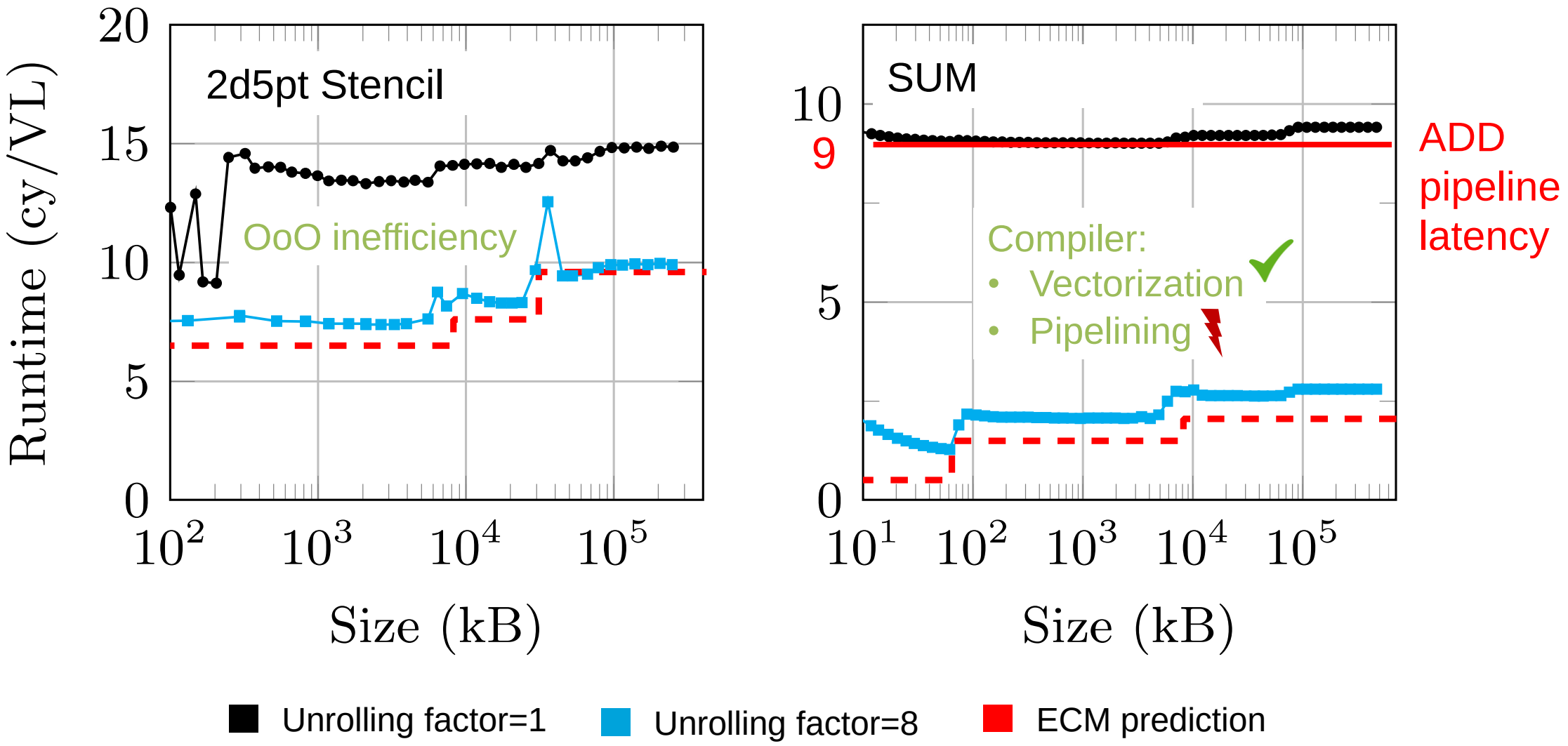
# Some things to take care of on FX700

# Data alignment is important



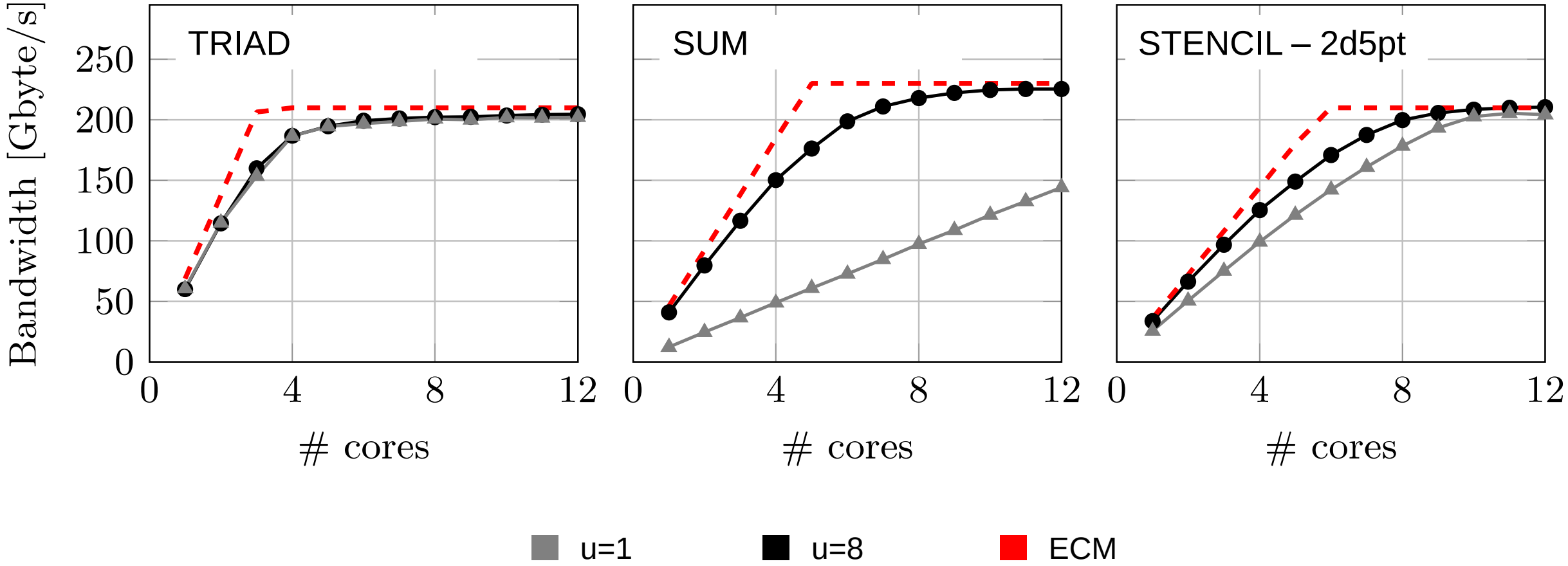

1024-bytes aligned  malloc  ECM model

# Unrolling is crucial



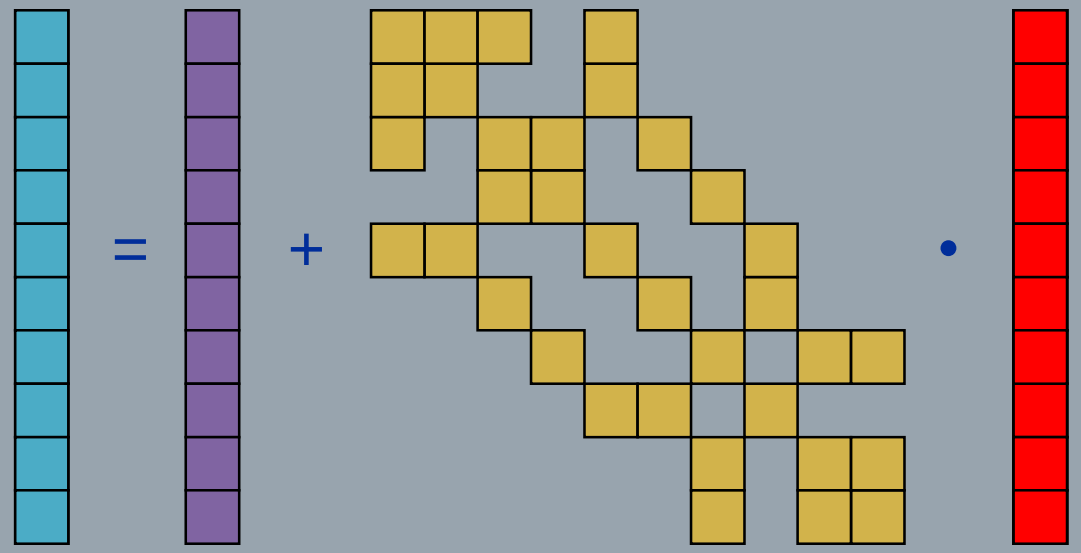
# Multicore saturation

# Multicore


Bandwidth [Gbyte/s]
250
200
150
100
50
0
TRIAD
SUM
STENCIL – 2d5pt
0
4
8
12
# cores
u=1
u=8
ECM

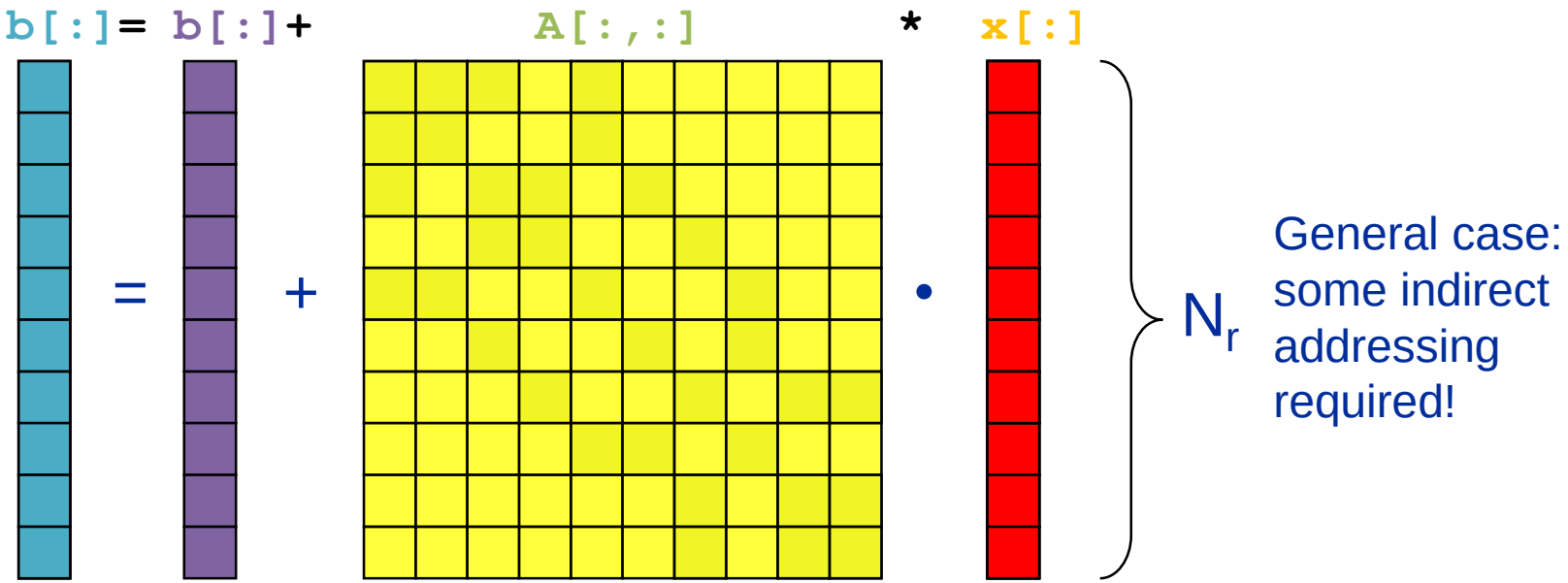
# SpMV

Sparse Matrix-Vector Multiplication



# SpMV

Sparse Matrix-Vector Multiplication (SpMV) : `b=Ax`




In Compressed Row Storage (CRS) format

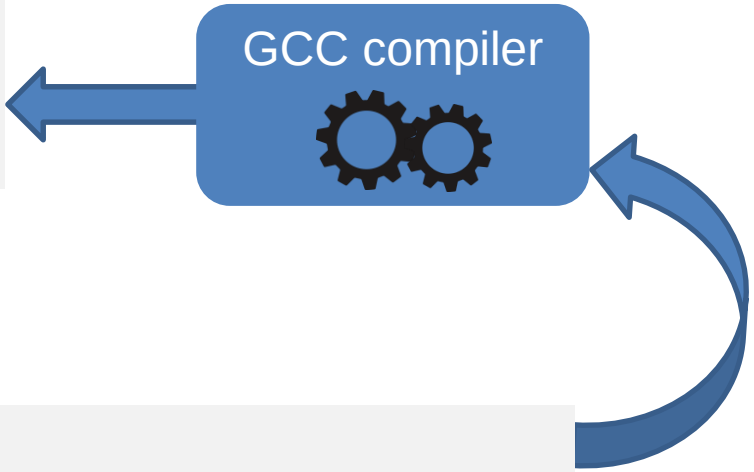
```
for i = 0:nrows-1 //Long outer loop
    for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

# SpMV

## Assembly of the short inner loop

```
.L6:
        ld1sw     z0.d, p0/z, [x17, x20, lsl 2]
        ld1d      z2.d, p0/z, [x18, x20, lsl 3]
        ld1d      z3.d, p0/z, [x30, z0.d, lsl 3]
        add       x20, x20, 8
        fmla      z1.d, p0/m, z3.d, z2.d
        whilelo   p0.d, x20, x14
        b.any     .L6

        faddv     d4, p1, z1.d
```




## In Compressed Row Storage (CRS) format

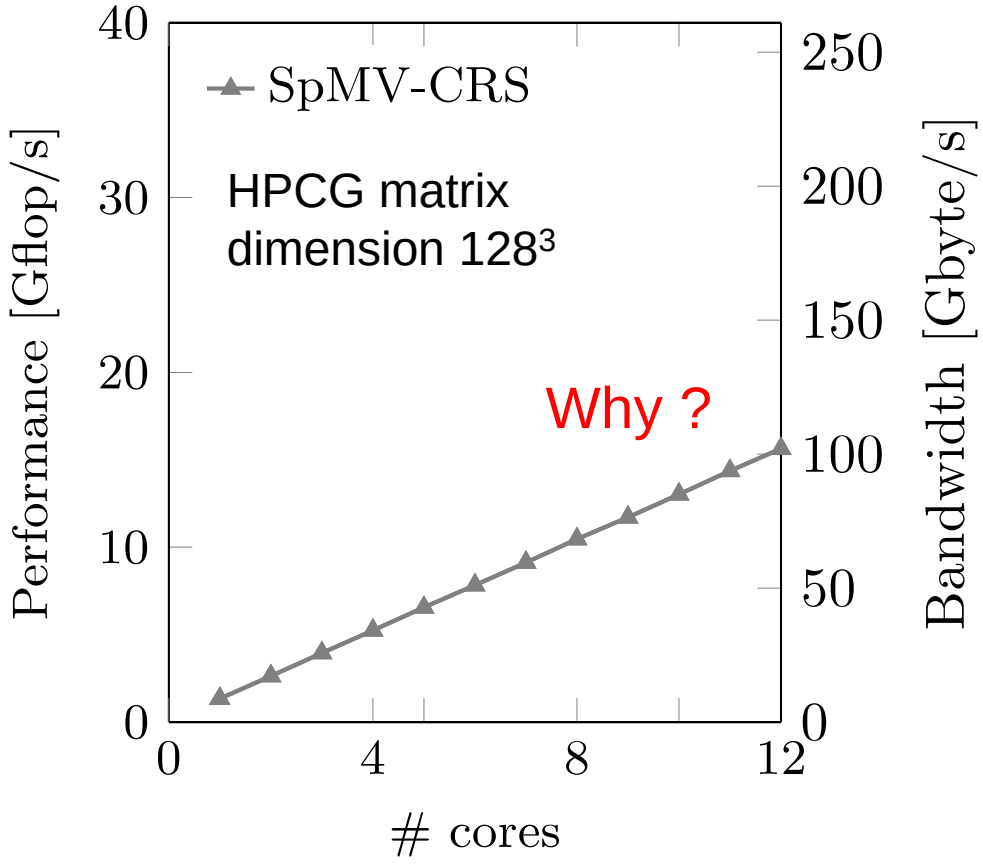
```
for i = 0:nrows-1 //Long outer loop
    for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

# SpMV

Assembly of the short inner-loop

```
.L6:
        ld1sw     z0.d, p0/z, [x17, x20, lsl 2]
        ld1d      z2.d, p0/z, [x18, x20, lsl 3]
        ld1d      z3.d, p0/z, [x30, z0.d, lsl 3]
        add       x20, x20, 8
        fmla      z1.d, p0/m, z3.d, z2.d
        whilelo   p0.d, x20, x14
        b.any     .L6

        faddv     d4, p1, z1.d
```




In Compressed Row Storage (CRS) format

```
for i = 0:nrows-1 //Long outer loop
    for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

# SpMV

## Assembly of the short inner-loop

```
.L6:
        ld1sw     z0.d, p0/z, [x17, x20, lsl 2]
        ld1d      z2.d, p0/z, [x18, x20, lsl 3]
        ld1d      z3.d, p0/z, [x30, z0.d, lsl 3]
        add       x20, x20, 8
        fmla      z1.d, p0/m, z3.d, z2.d
        whilelo   p0.d, x20, x14
        b.any     .L6

        faddv     d4, p1, z1.d
```

FMA: Update **z1.d**

Latency: 9 cycles

Loop length : 27
**HPCG matrix**

Horizontal add of 512-bit register

Throughput = 11.5

ECM model predicts maximum bandwidth of 160 GB/s[1]

→ No saturation ☹

## In Compressed Row Storage (CRS) format

```
for i = 0:nrows-1 //Long outer loop
    for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

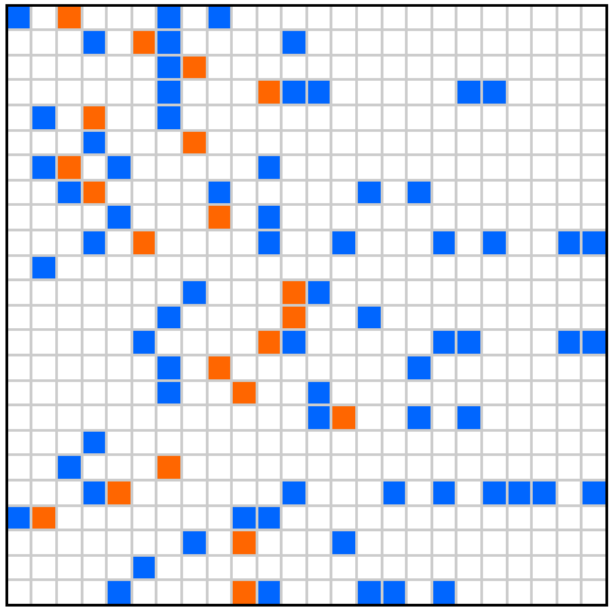
[1]DOI:10.1109/PMBS51919.2020.00006

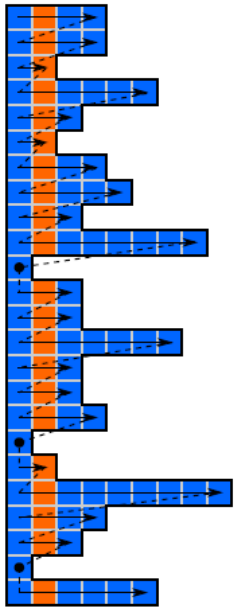
# CRS → SELL-C-σ

Change data storage format

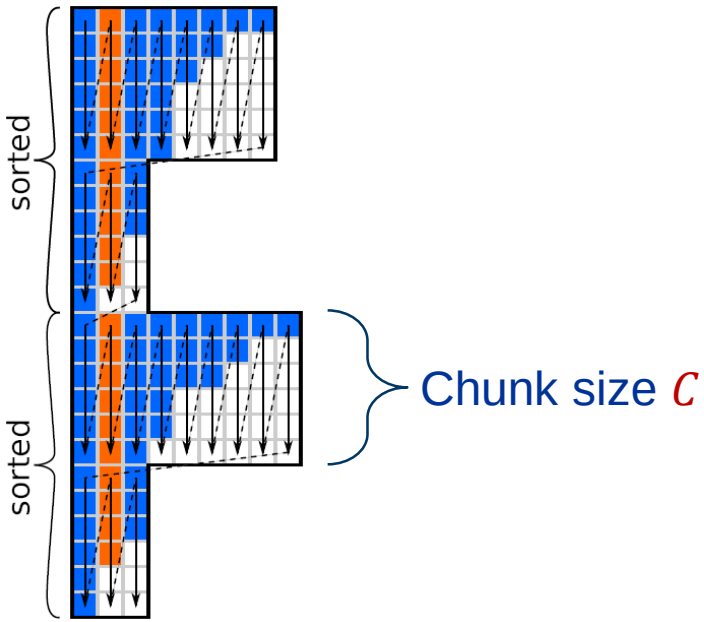
# SpMV – SELL-C-σ

Vector-friendly SpMV data format/kernel required for A64FX → SELL-C-σ[1]





CRS




SELL-C-σ

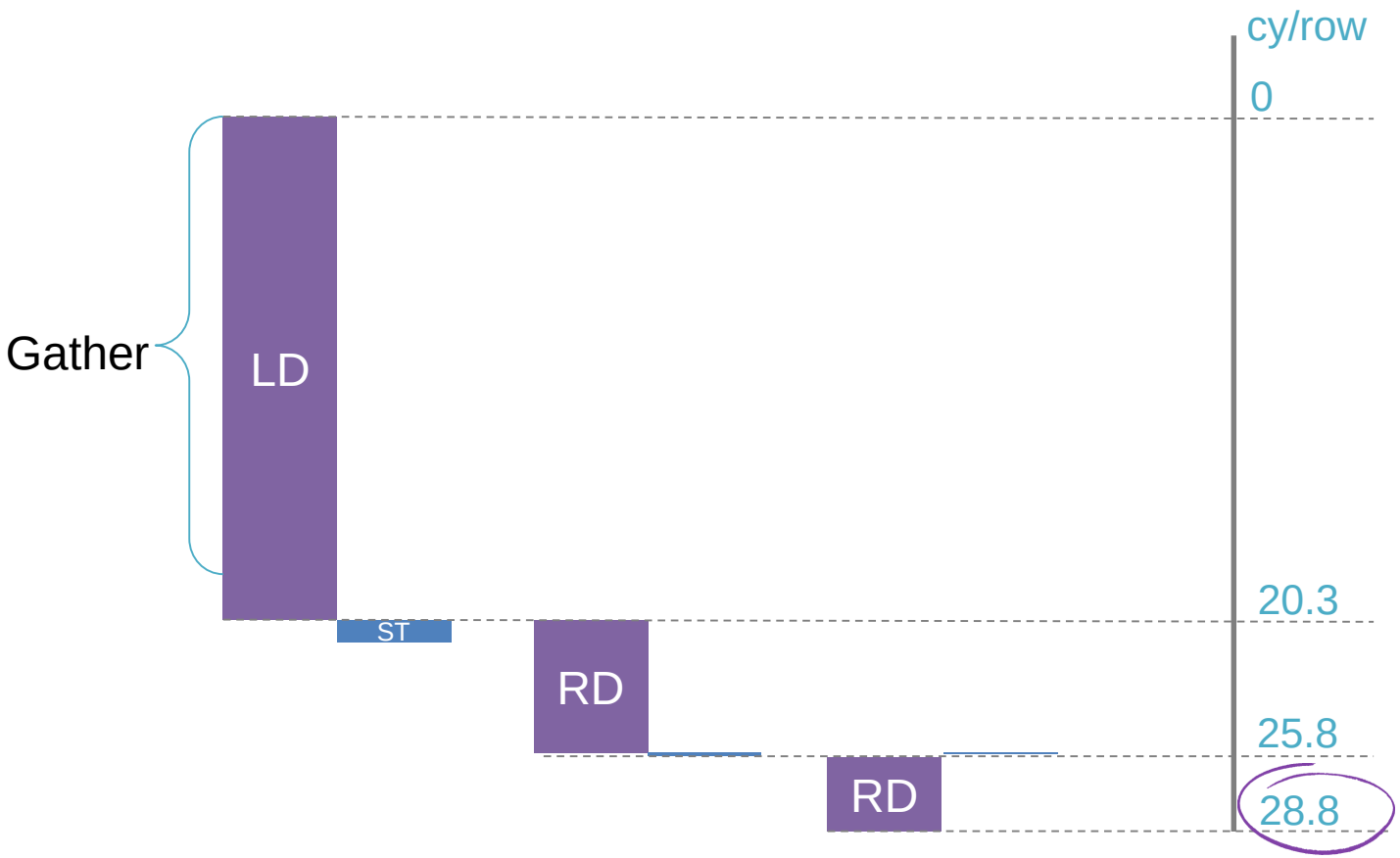
Benefits:

- Vectorization and unrolling along chunk size (C) → long loop and tunable
- No costly horizontal-add (faddv)
- No loss of vector efficiency

[1]M. Kreutzer et al., A Unified Sparse Matrix Data Format For Efficient General Sparse Matrix-vector Multiplication On Modern Processors With Wide SIMD Units, SIAM SISC 2014, DOI: 10.1137/130930352

# ECM model for SELL-C-σ

HPCG matrix, dimension $128^3$

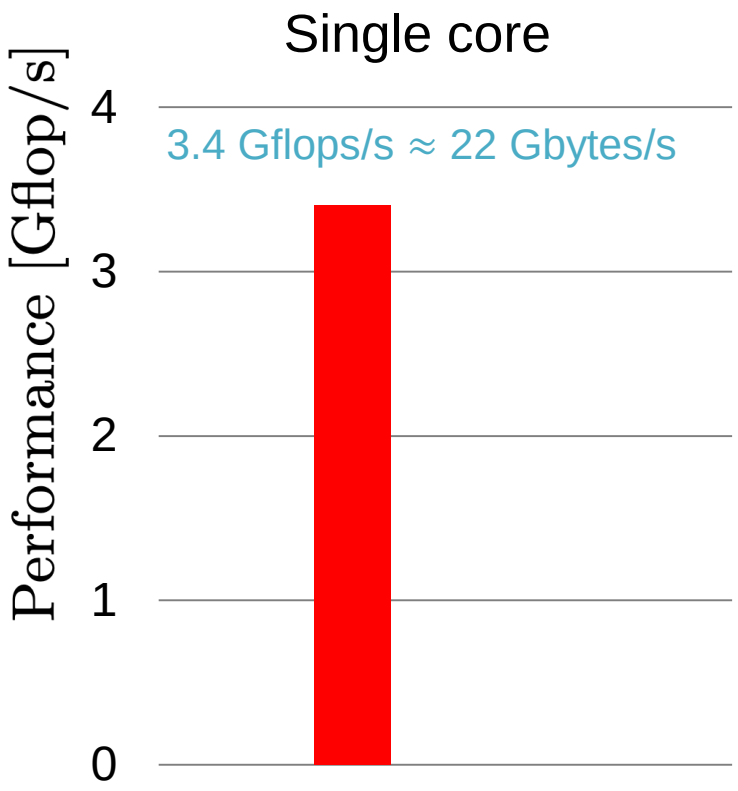


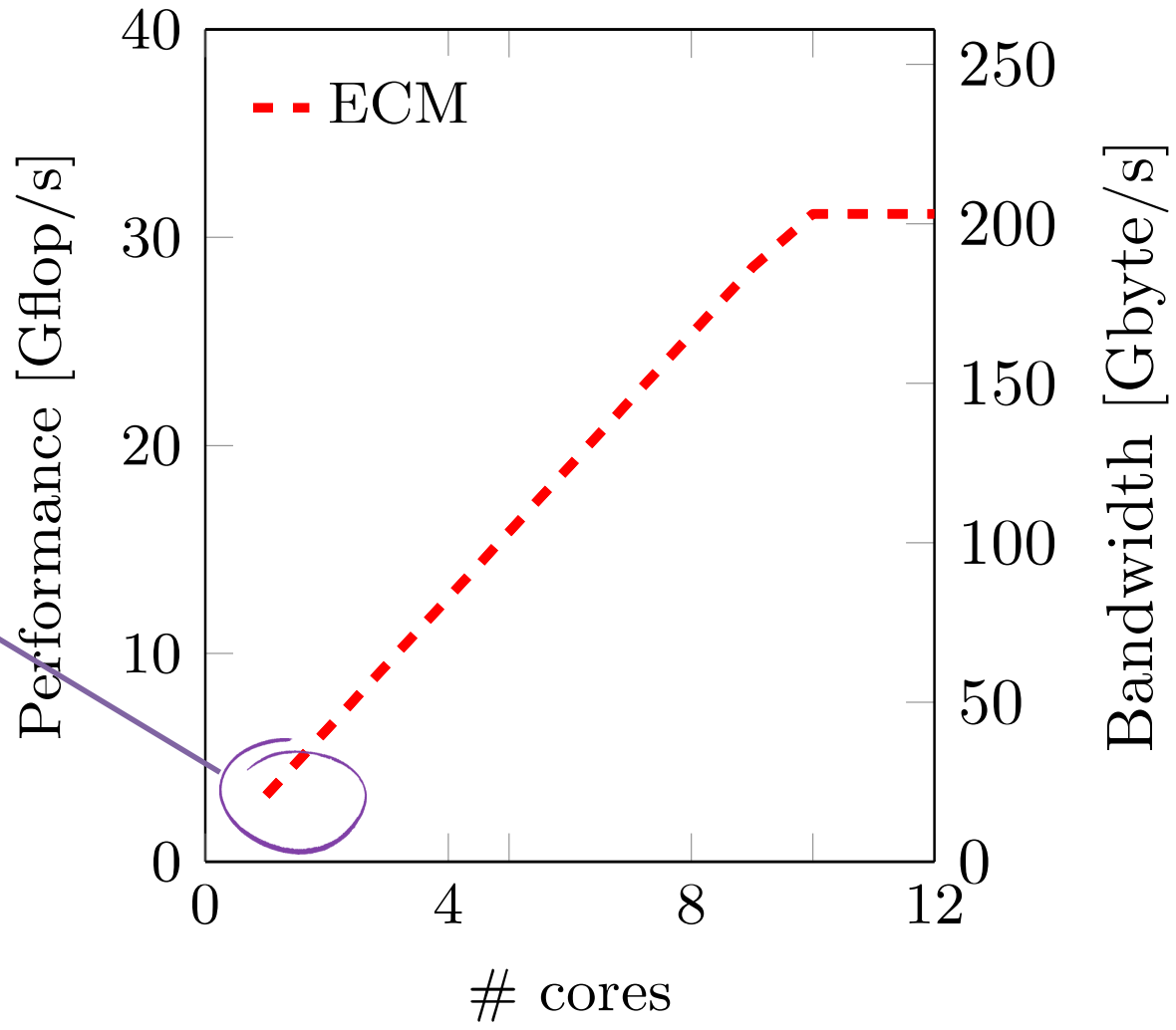

= 3.4 Gflops/s → 22 Gbyte/s

# ECM model for SELL-C-σ

HPCG matrix, dimension $128^3$

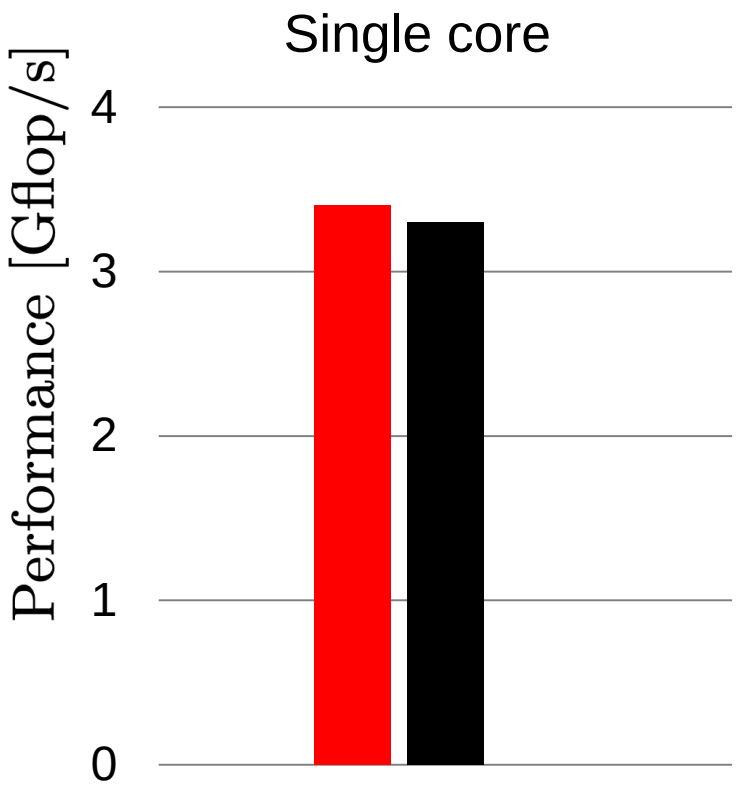
Can we saturate now ?



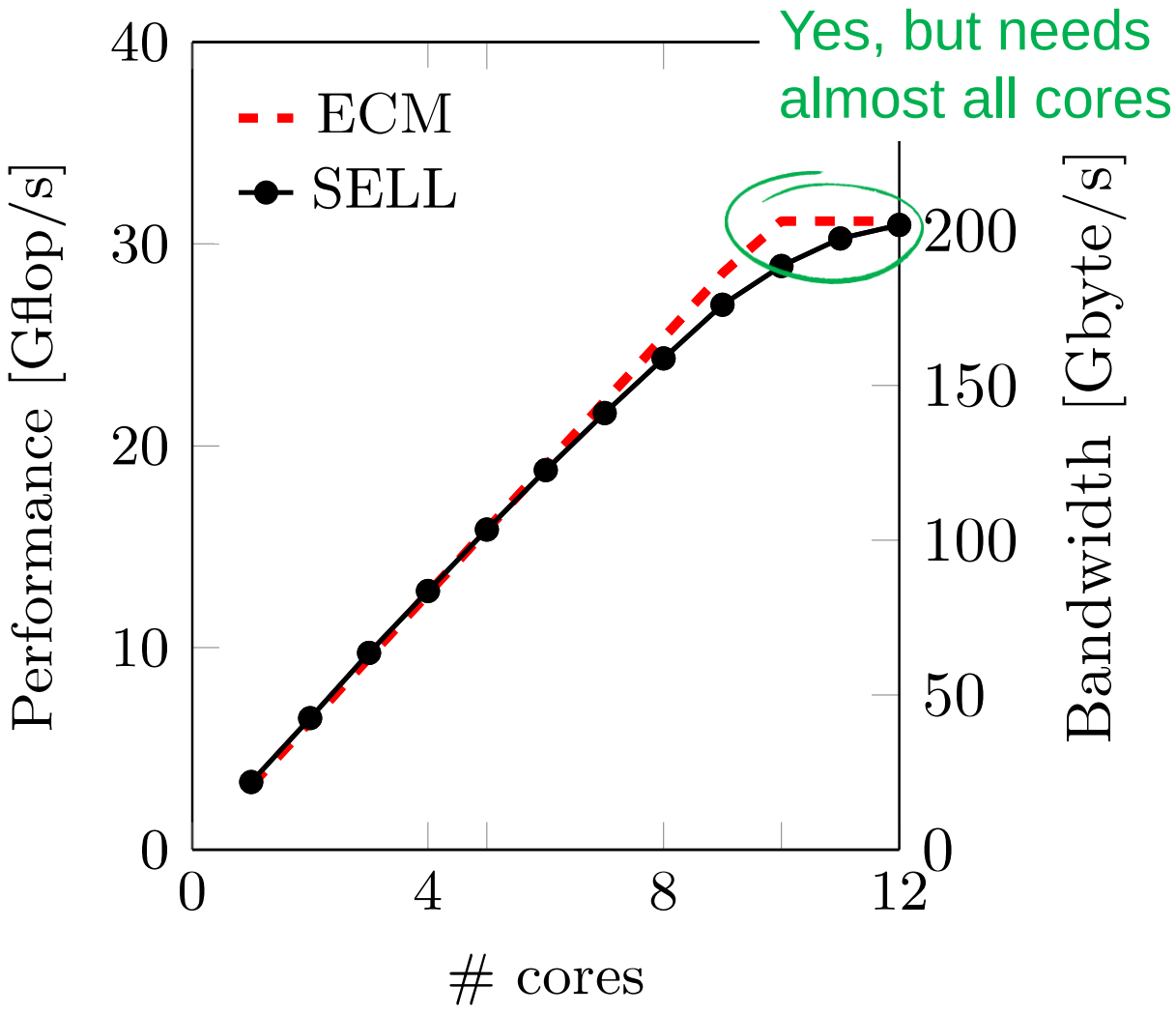



# ECM model for SELL-C-σ

HPCG matrix, dimension $128^3$

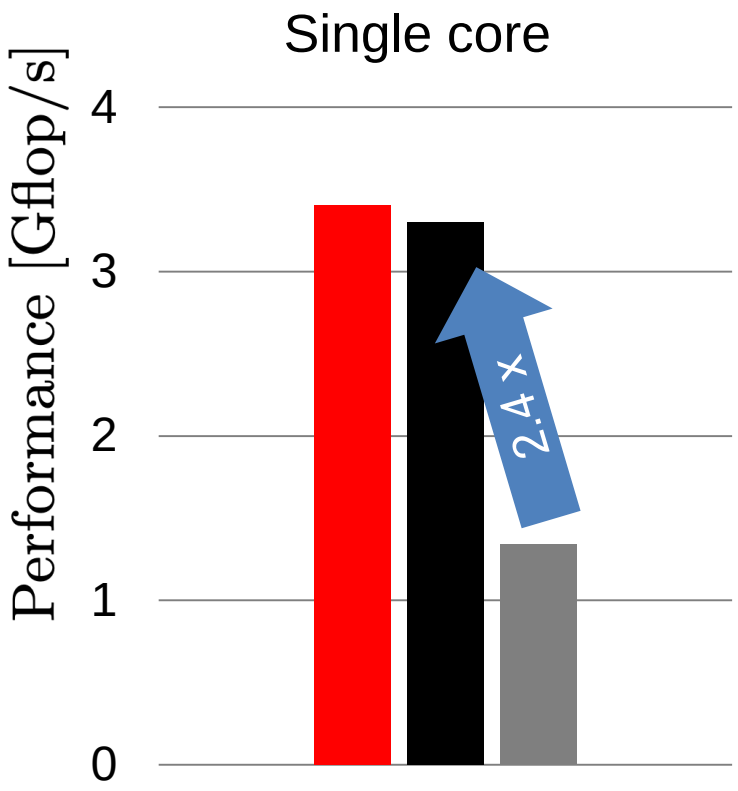


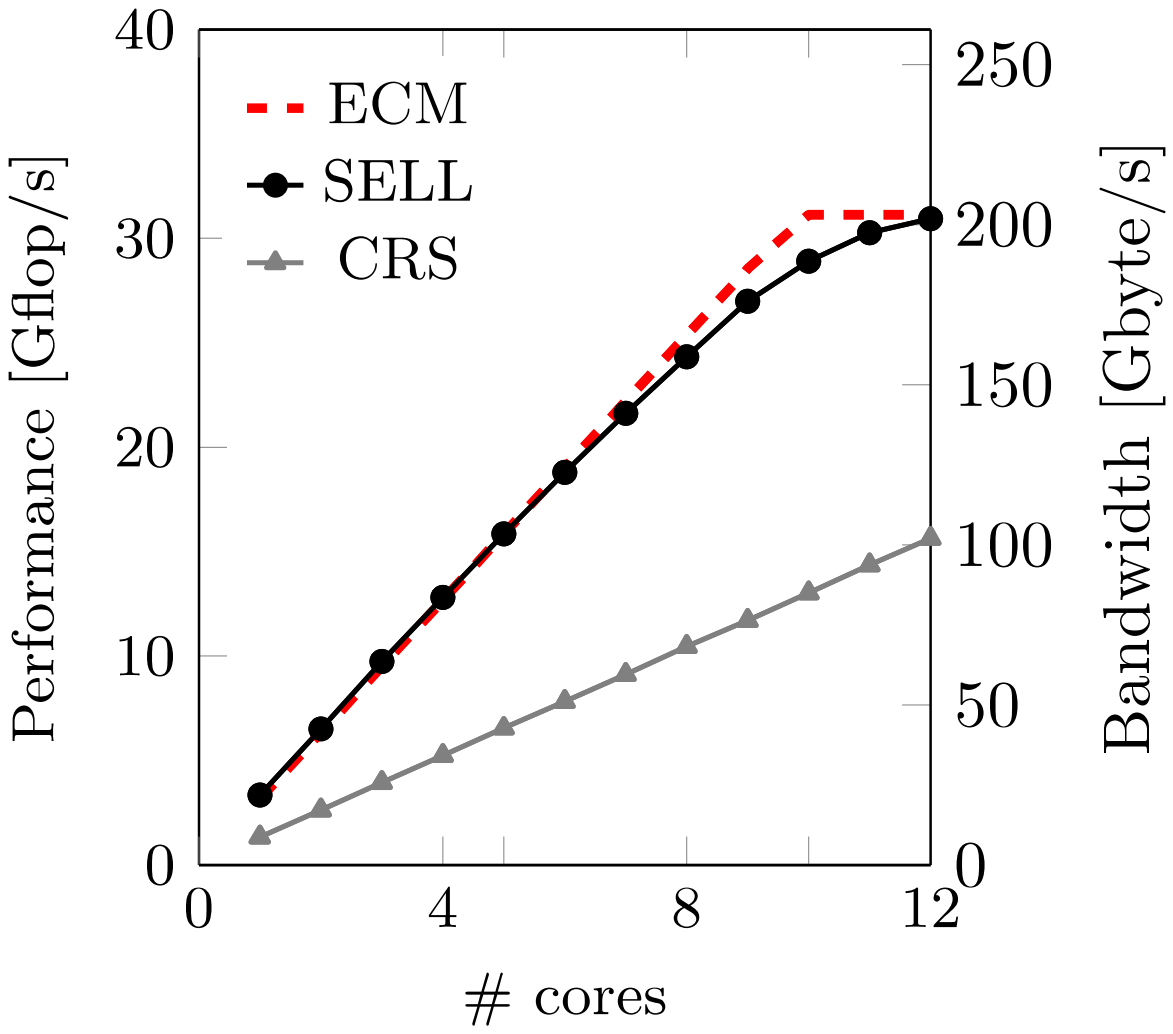

Can we saturate now ?



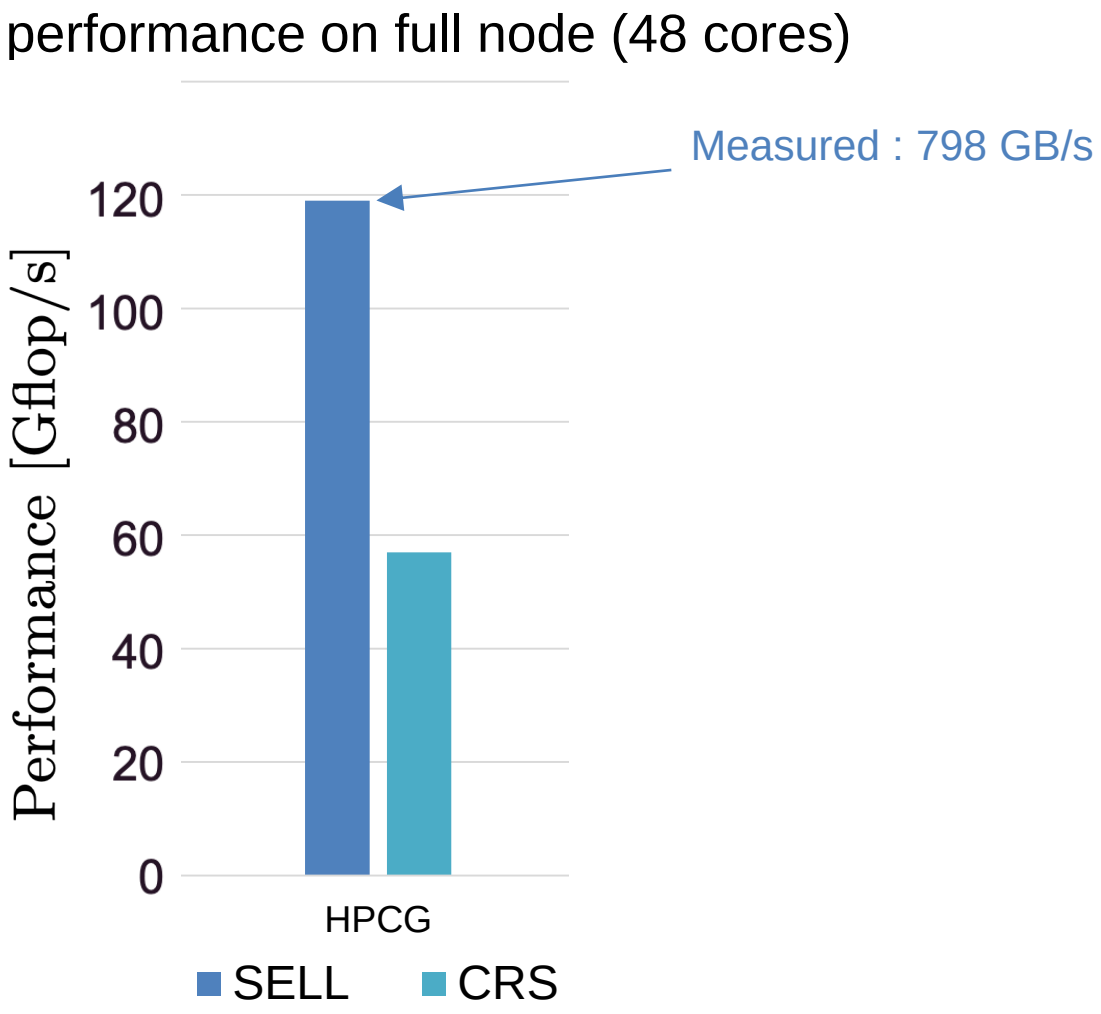
# ECM model for SELL-C-σ

HPCG matrix, dimension $128^3$

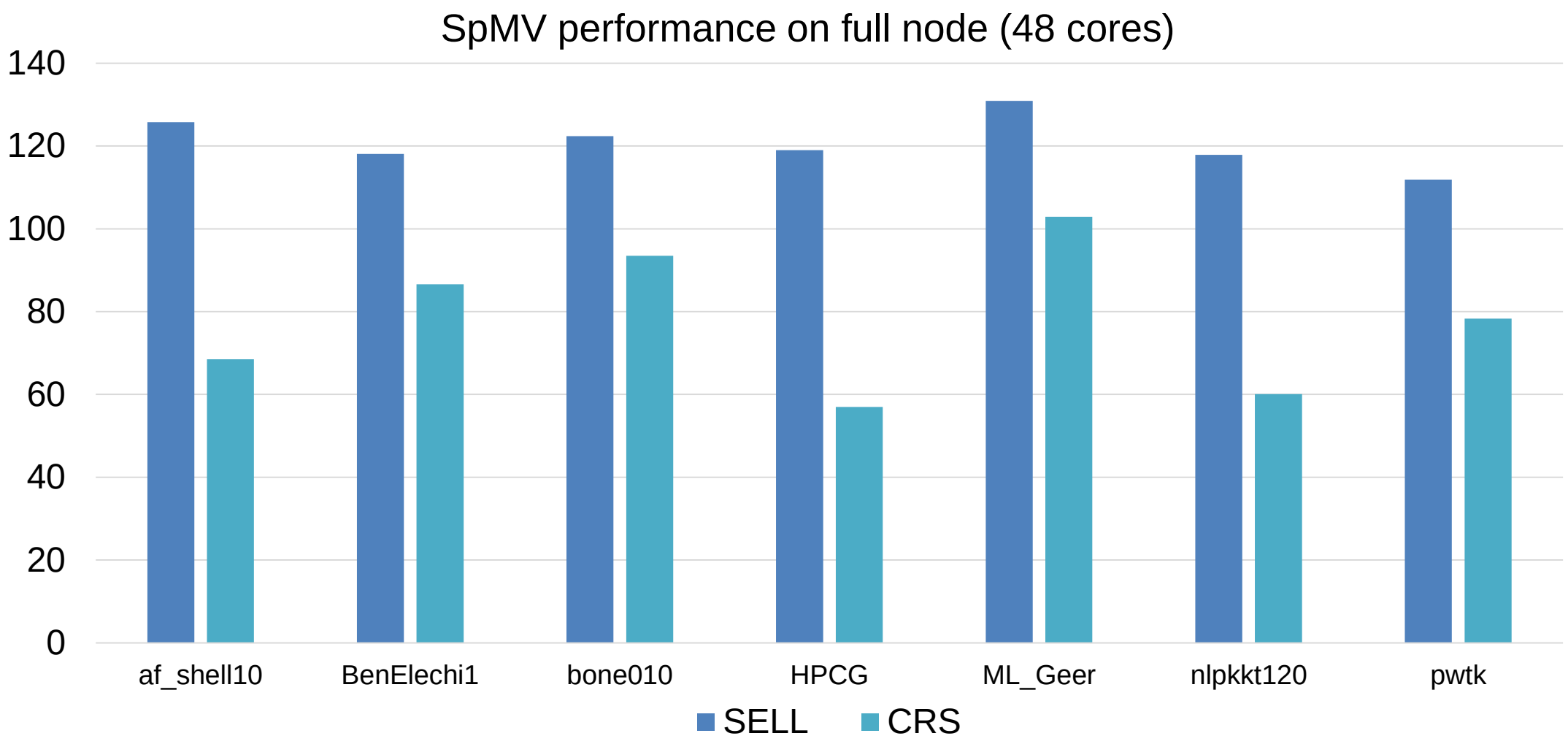


# SpMV performance

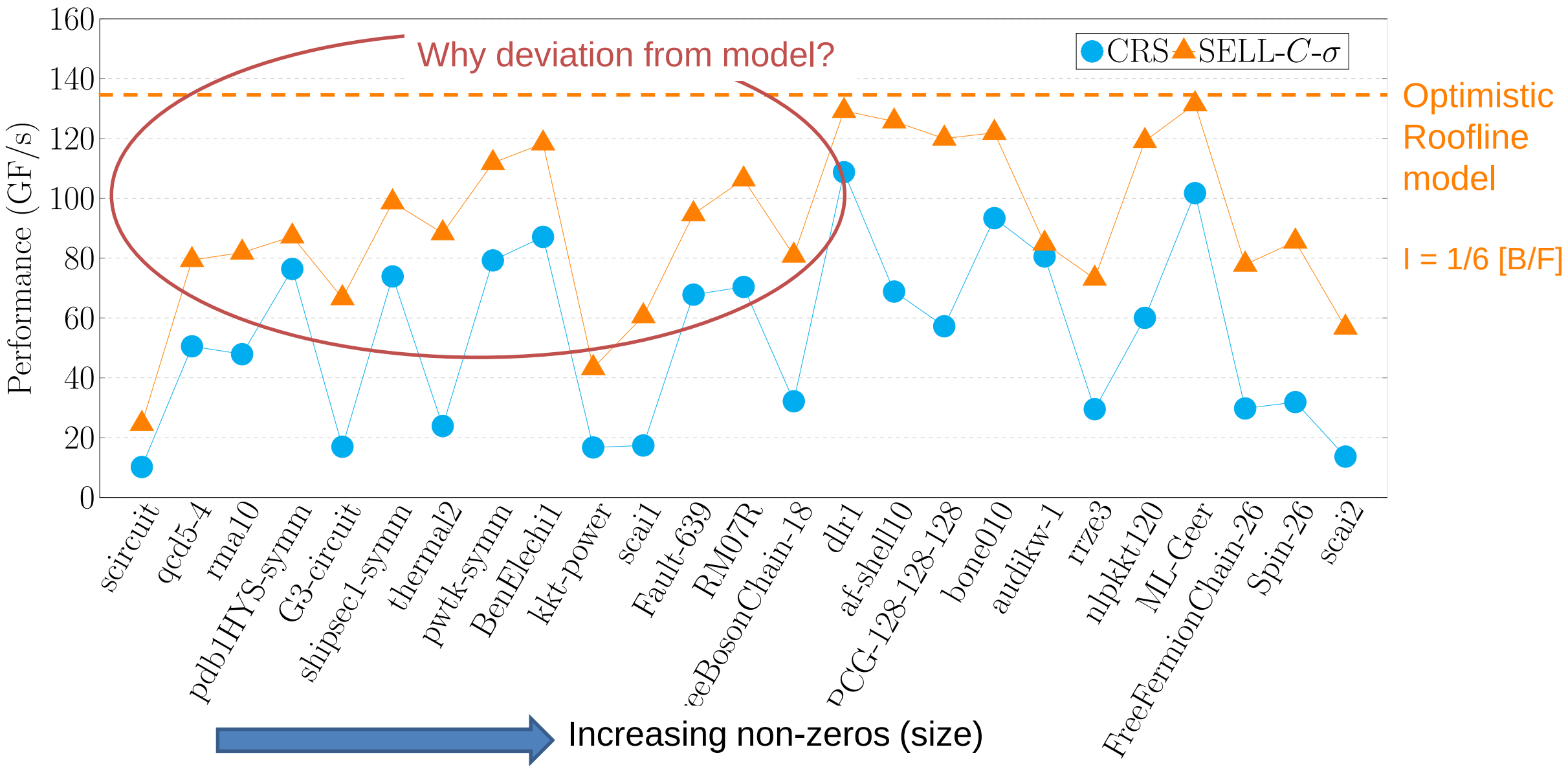


# SpMV performance

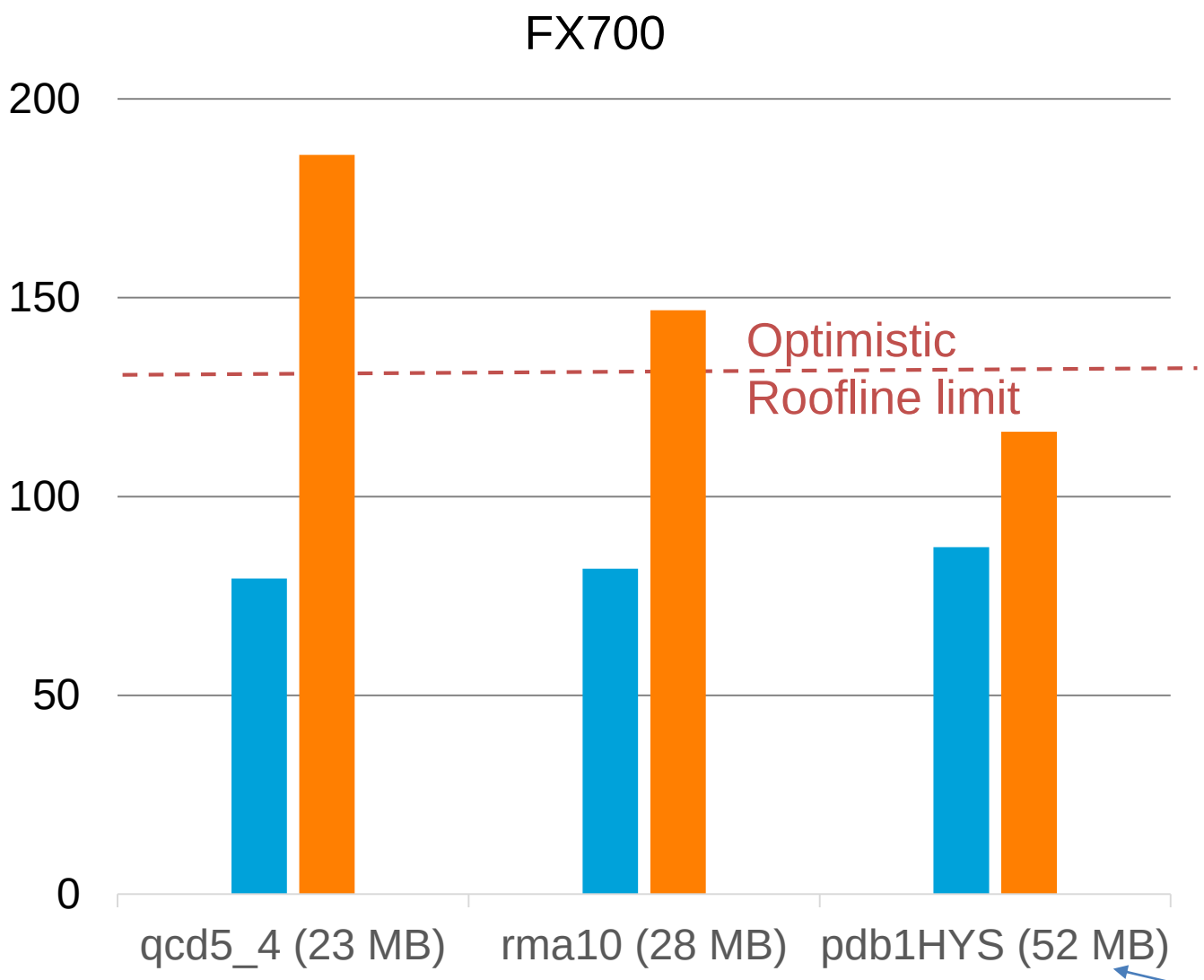


# SpMV performance of FX700

# CRS vs SELL-C-σ : 1 Node



# BARRIER pain and SpMV




Data-set size

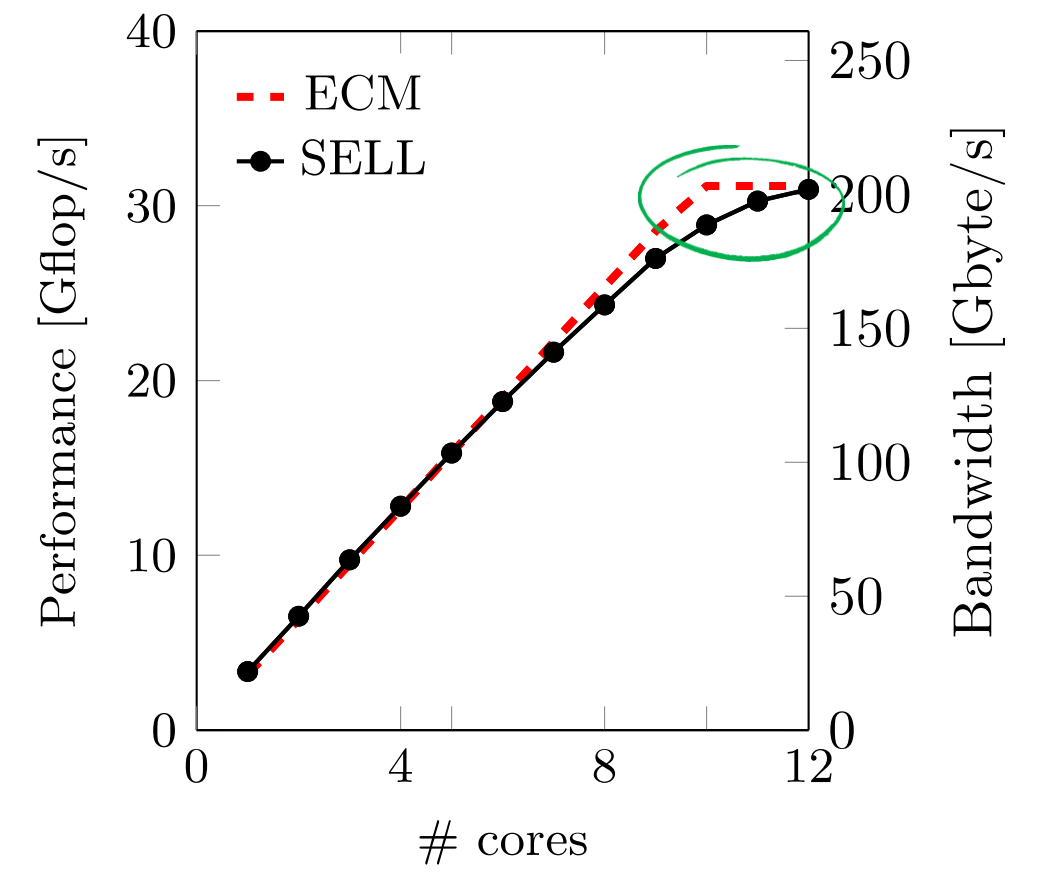
For small matrices, software barrier overhead can be hazardous.

Hardware barrier might help in these cases.

with barrier

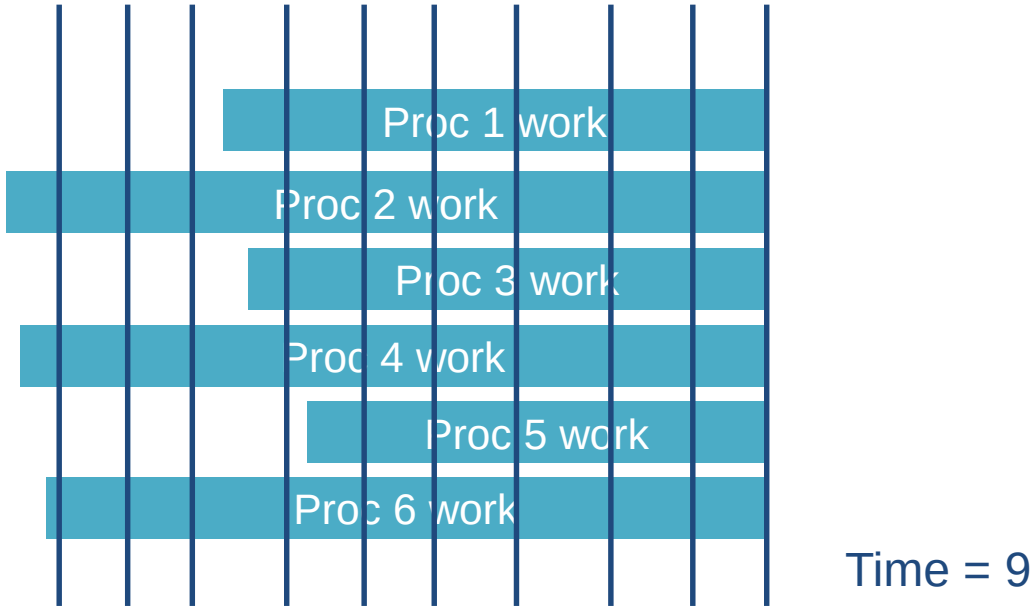
without barrier (wrong numerics)

# Consequences of “last core saturation”



# Load balancing

Early memory saturation absorbs some load imbalance




Case 1: Early saturation
Memory bandwidth saturated
with 3 cores

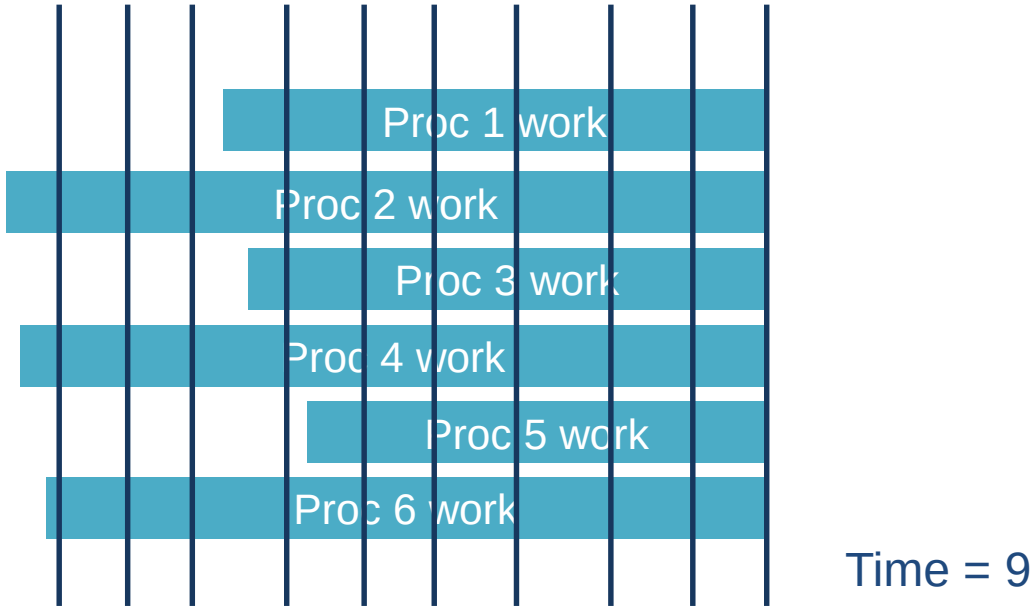
Active cores : 6

At least 3 cores are active most of the time

➔ full bandwidth ☺

# Load balancing

Late saturation → Sensitive to load imbalance and irregularities



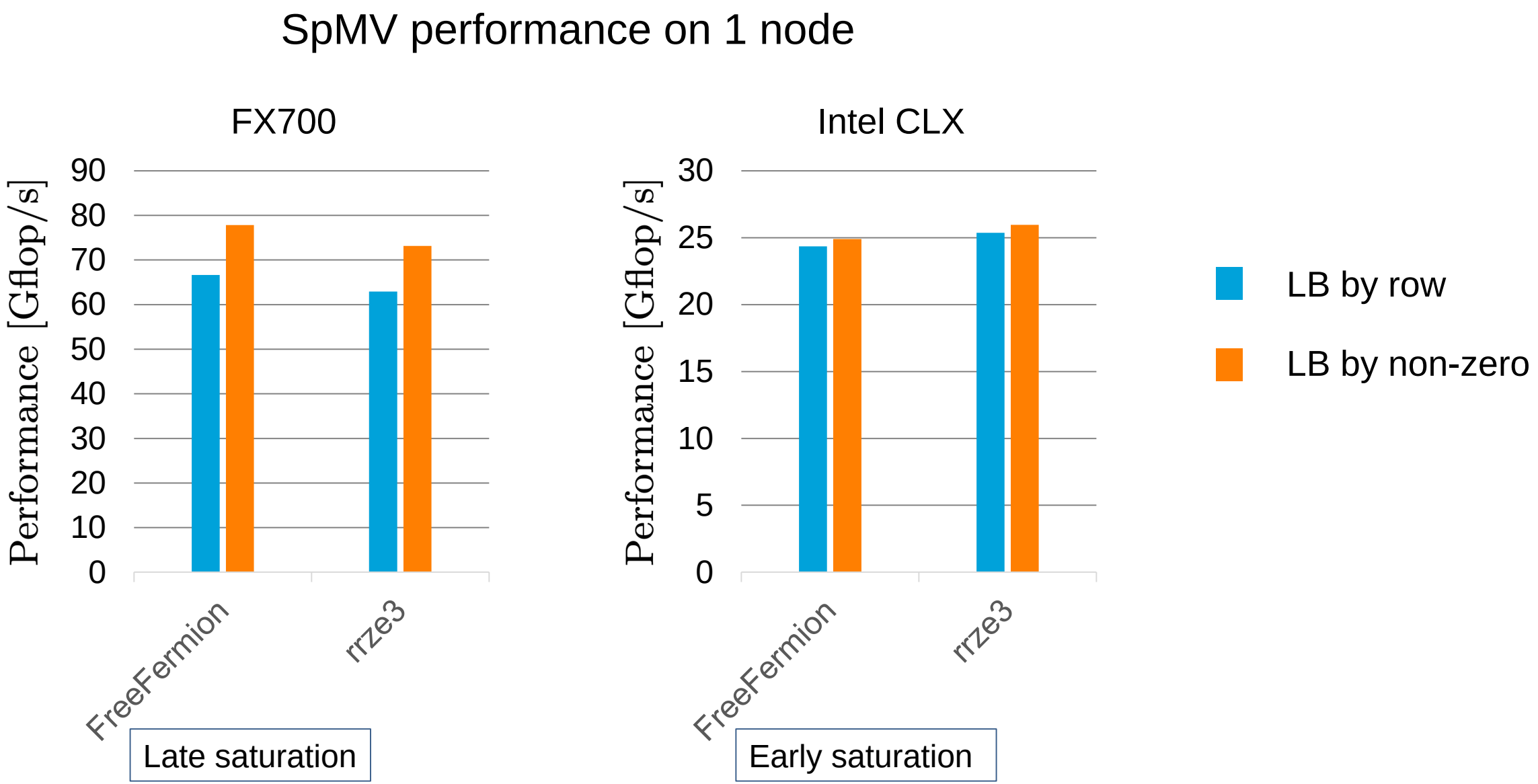

Case 2: Late saturation
Memory bandwidth saturated
with 5 cores

Active cores : 6

Less than 5 cores active

→bandwidth reduction

# Load balancing (LB)



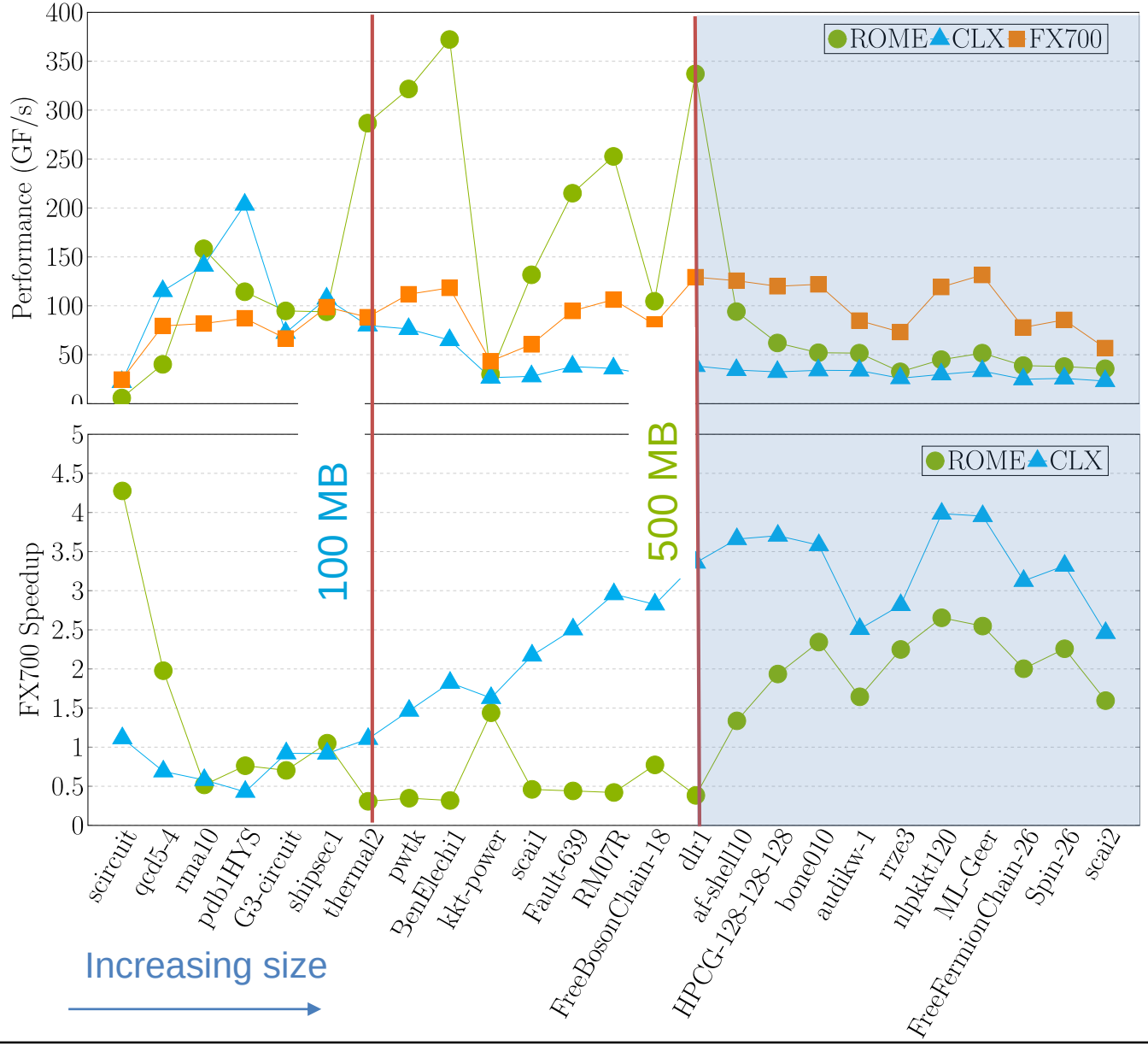
# Comparison with other CPUs and the V100 GPU

**1 Node FX700 : 1 x A64FX (48 c @ 1.8 GHz)**

**1 Node CLX : 2 x Intel Xeon Platinum 8280 (28 c @ 2.7 GHz )**

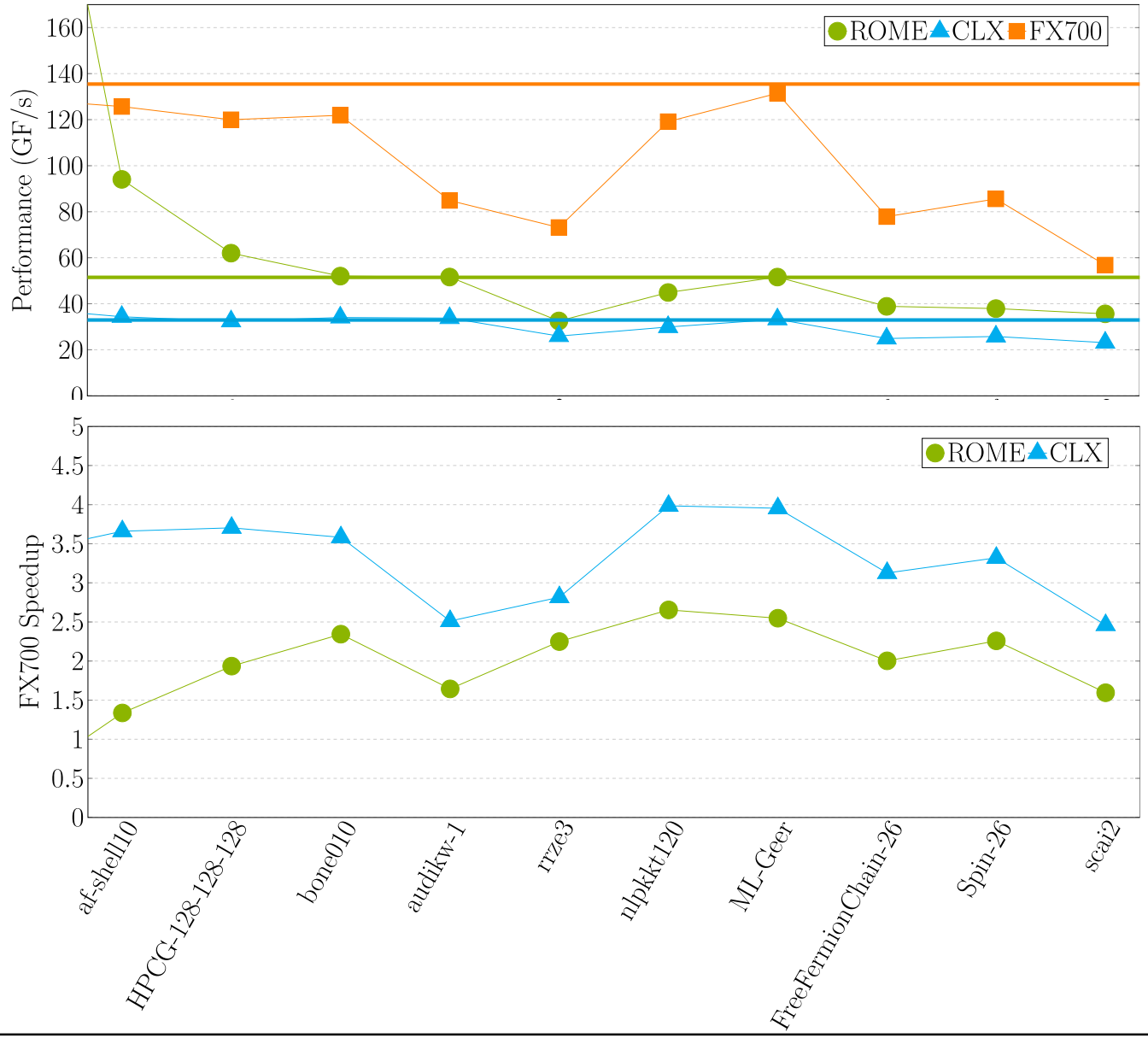
**1 Node ROME : 2 x AMD EPYC 7742 (64 c @ 2.25 GHz)**

# CPU comparison: SpMV




| | LLC size |
|---|---|
| Fujitsu FX700 | 32 MiB |
| AMD ROME 7742 | 512 MiB |
| Intel Cascade Lake 8280 | 77 MiB (+ 56 MiB L2) |

# CPU comparison – memory-bound cases

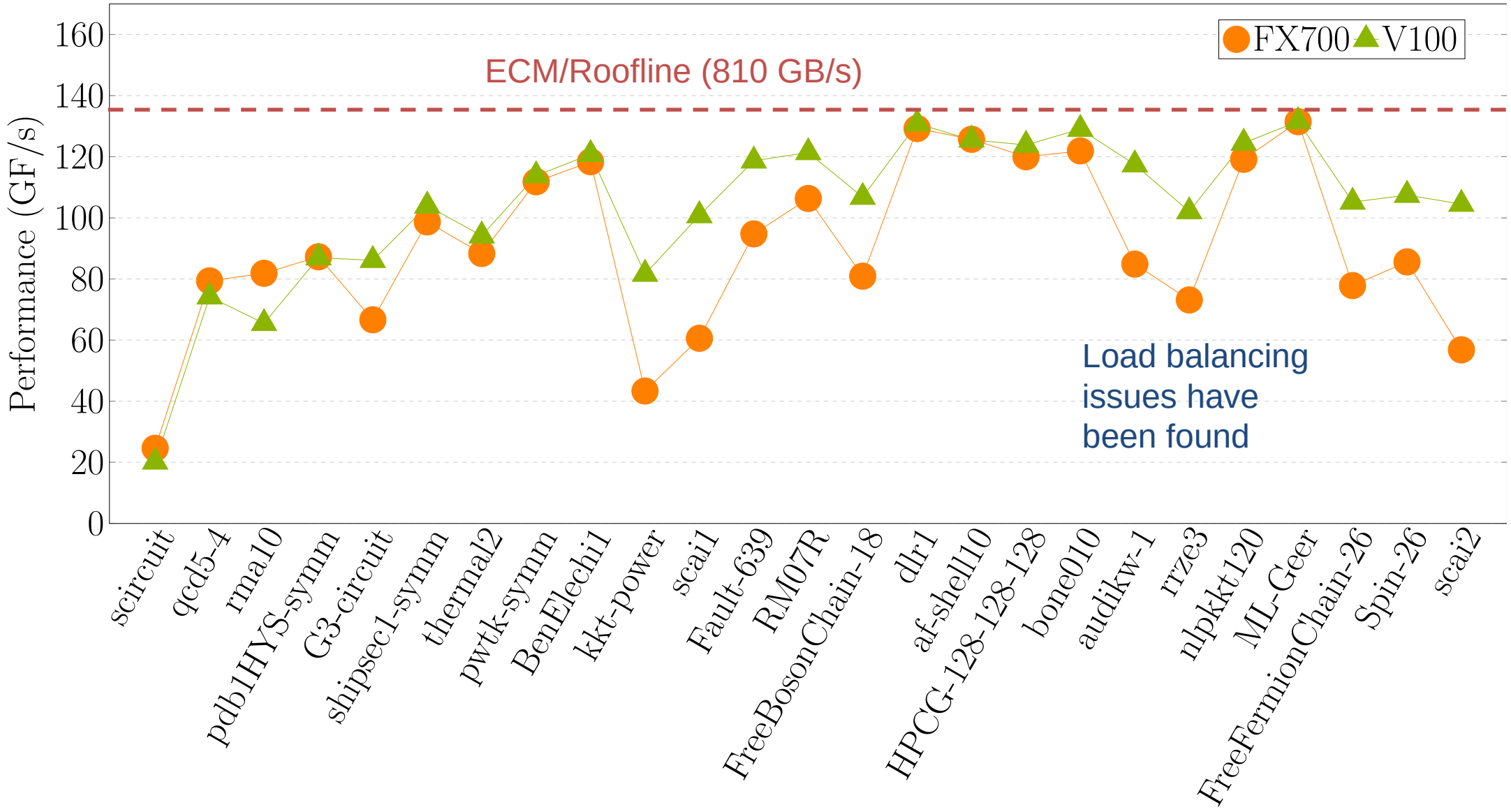



ECM/ Roofline limits

| | MemBW |
|---|---|
| Fujitsu FX700 | 810 GB/s |
| AMD ROME 7742 | 320 GB/s |
| Intel Cascade Lake 8280 | 220 GB/s |

2× and 3× speedup over ROME and CLX.

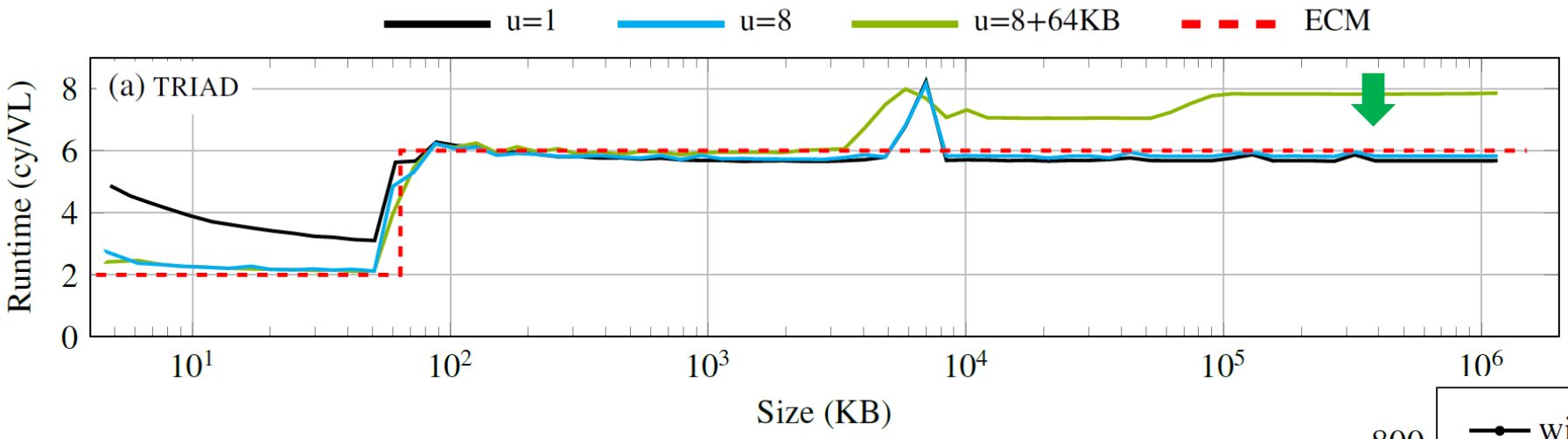
# GPU comparison: SpMV



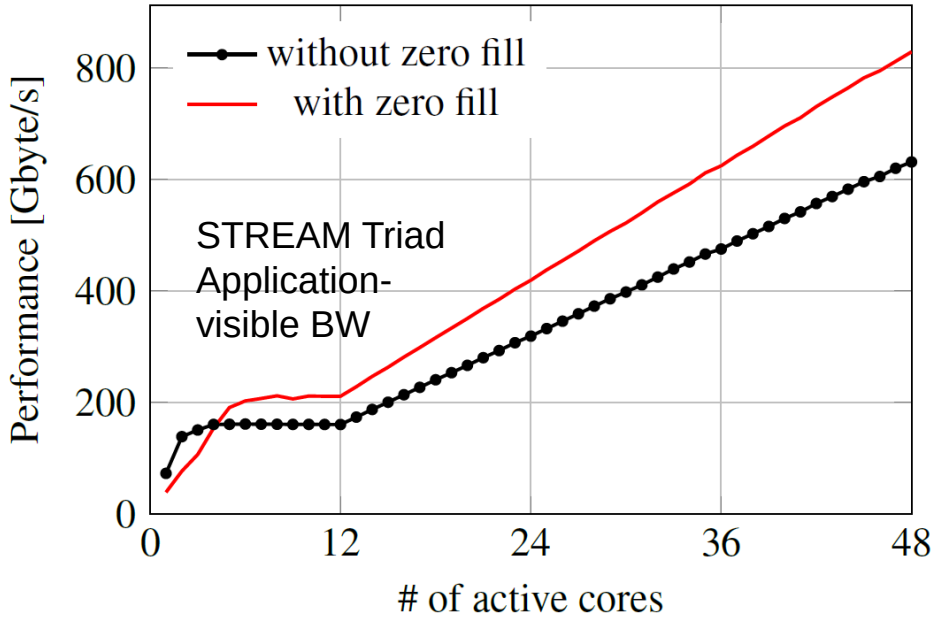
# Outlook: Advanced features, FX1000

# Outlook (1)

Large pages have a positive effect on L2 overlapping (→ updated ECM model)

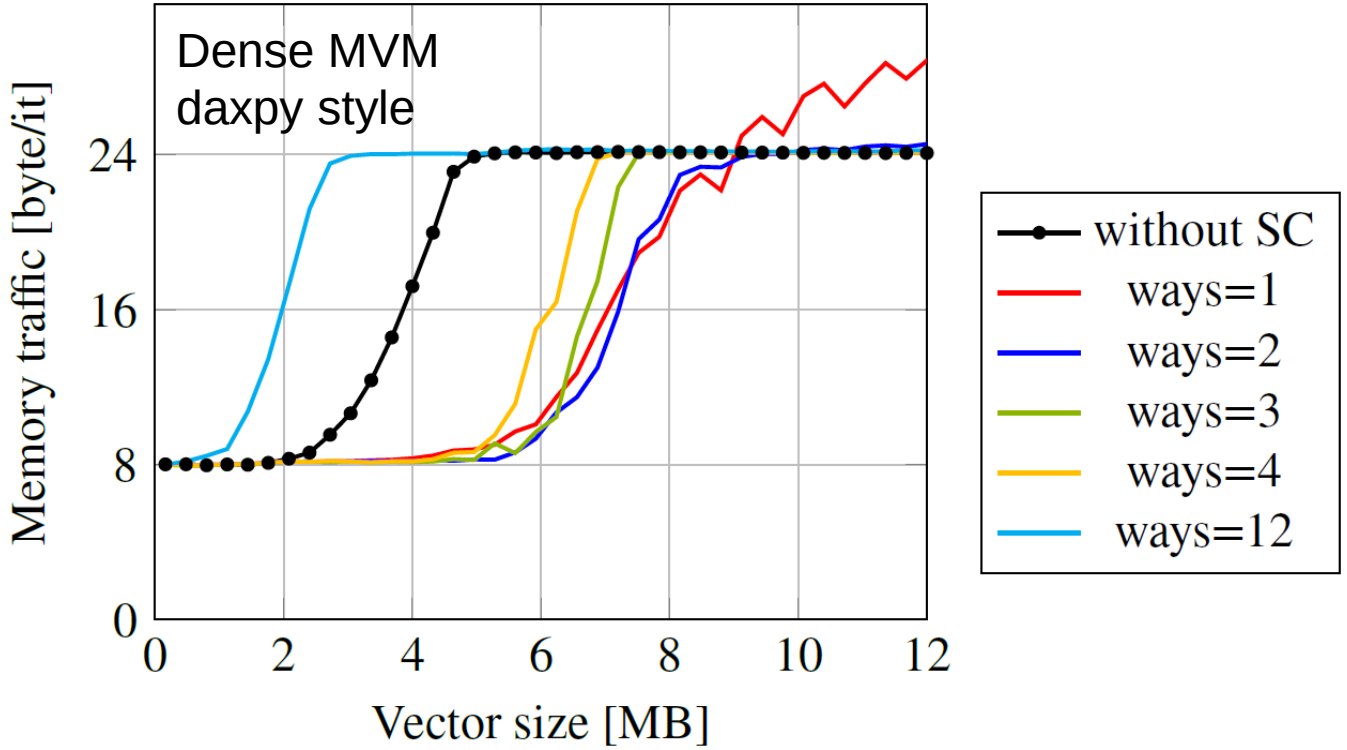


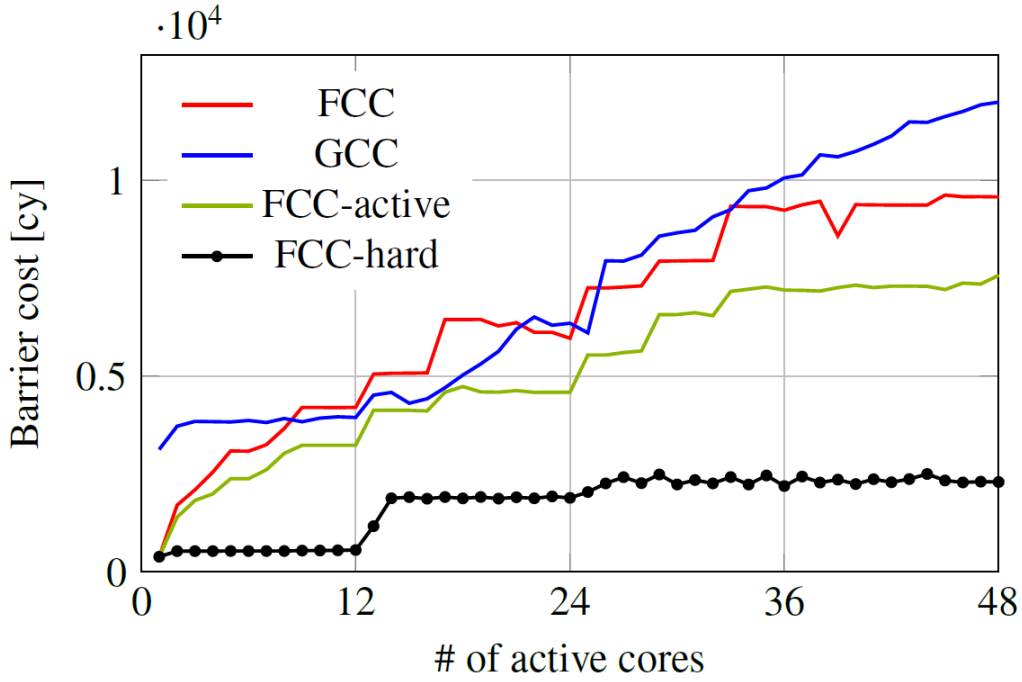

Cache line zero instructions can avoid write-allocate transfers:



# Outlook (2)

Sector cache can restrict the use of cache ways for data structures




Hardware barrier can reduce the impact of barrier synchronization

# Summary

## Conclusions

- ECM model constructed for single-core performance of FX700/A64FX
- Partially overlapping memory hierarchy → high single-core memory bandwidth (even more so with large pages)
- SpMV with large matrices: FX700 is 2× and 3× faster than top-bin AMD ROME and Intel CLX
- Competitive with NVIDIA V100 for SpMV

## Lessons learned

- Proper single-core optimizations required to hide long floating point latency and inefficiencies in OoO

  → Open-Source compilers have headroom for improvement

- SpMV BW saturation with SELL-C-$\sigma$ format
- Proper load balancing is crucial
- Hardware barrier can reduce synchronization cost

# Thank you. Questions ?

## Visit our YouTube channel:

**http://tiny.cc/YT-RRZE-HPC**

KONWIHR

HLRS
High-Performance Computing Center | Stuttgart

SFB TRR55