SC20

Everywhere we are | more than hpc.

[12 Nov 2020] • [AoE]

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

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

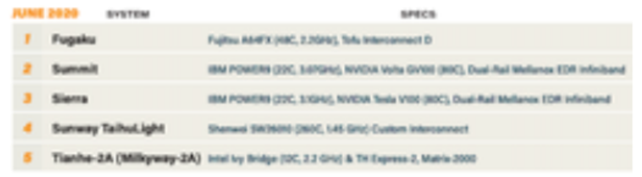
A64FX

NEWS

Japan Captures TOP500 Crown with Arm-Powered Supercomputer

June 22, 2020

FRANKFURT, Germany; BERKELEY, Calif.; and KNOXVILLE, Tenn.—The 55th edition of the TOP500 saw some significant additions to the list, spearheaded by a new number one system from Japan. The latest rankings also reflect a steady growth in aggregate performance and power efficiency.

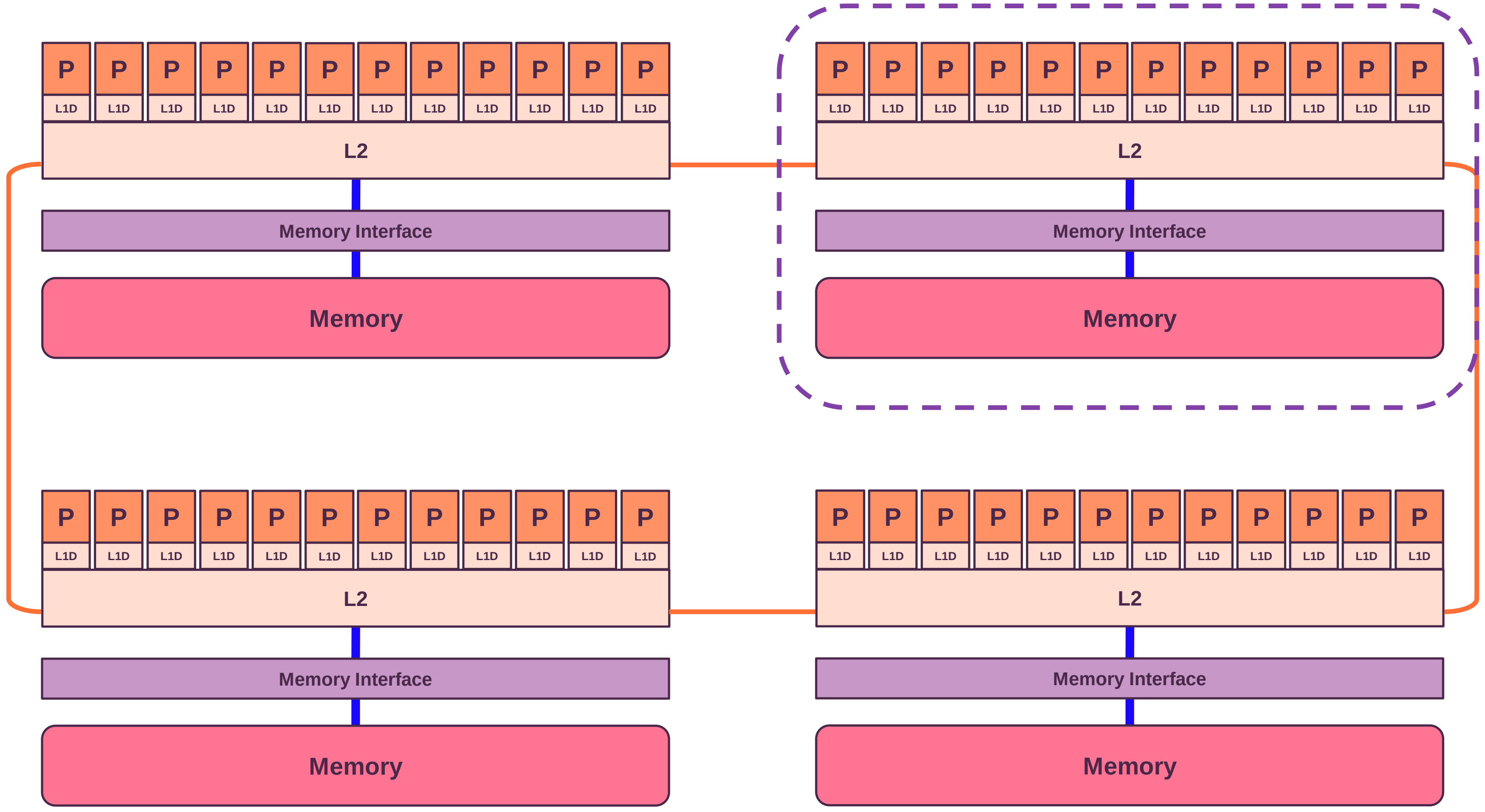


JUNE 2020	SYSTEM	SPECS
1	Fugaku	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D
2	Summit	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband
3	Sierra	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband
4	Sunway TaihuLight	Shenwei SW26010 (260C, 1.45 GHz) Custom Interconnect
5	Tianhe-2A (Milkyway-2A)	Intel Ivy Bridge (12C, 2.2 GHz) & TH Express-2, Matrix-2000

The new top system, Fugaku, turned in a High Performance Linpack (HPL) result of 415.5 petaflops, besting the now second-place Summit system by a factor of 2.8x. Fugaku, is powered by Fujitsu's 48-core A64FX SoC, becoming the first number one system on the list to be powered by ARM processors. In single or further reduced precision, which are often used in machine learning and AI applications, Fugaku's peak performance is over 1,000 petaflops (1 exaflops). The new system is installed at RIKEN Center for Computational Science (R-CCS) in Kobe, Japan.

Source : https://www.top500.org/news/japan-captures-top500-crown-arm-powered-supercomputer/

A64FX – FX700

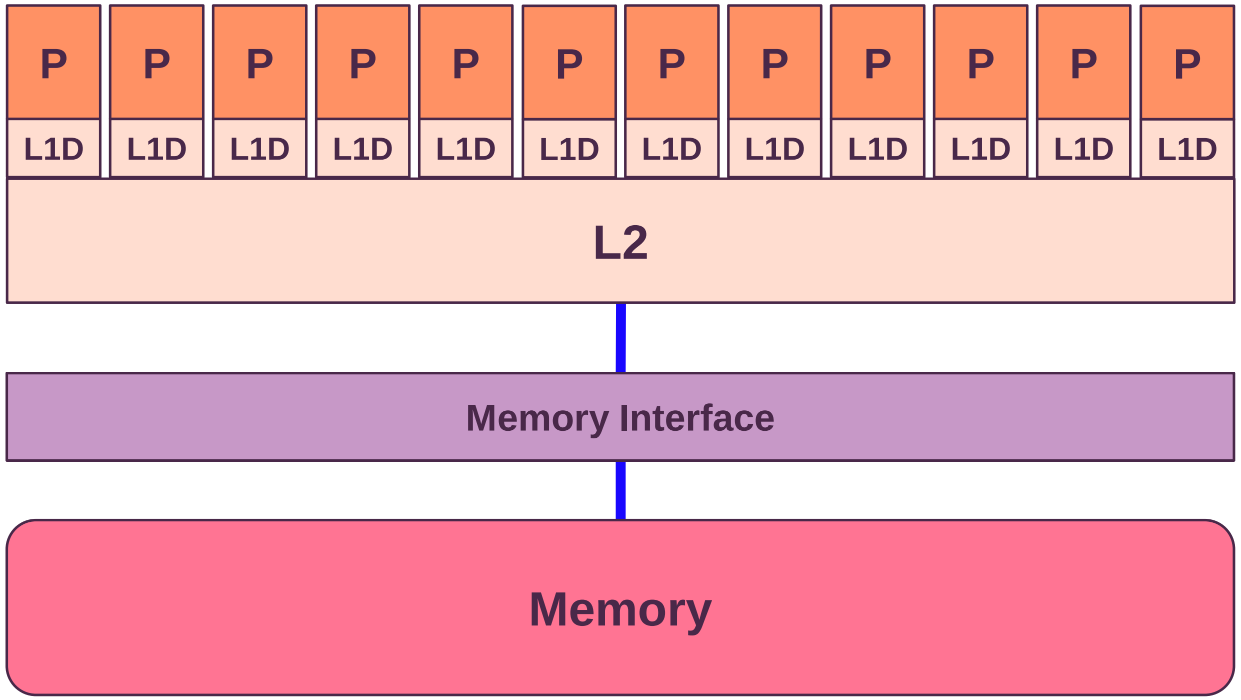


1 CMG

4 CMGs
per node

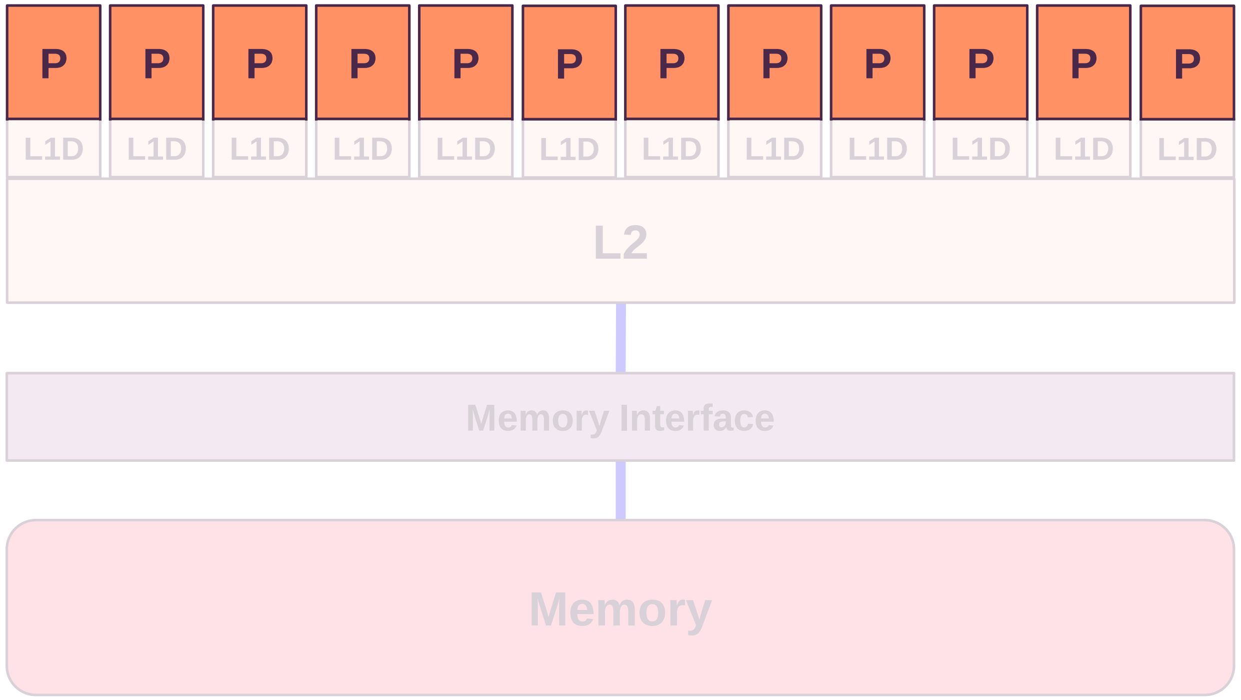
48 cores
per node

A64FX – FX700



1 CMG

A64FX – FX700



1 CMG

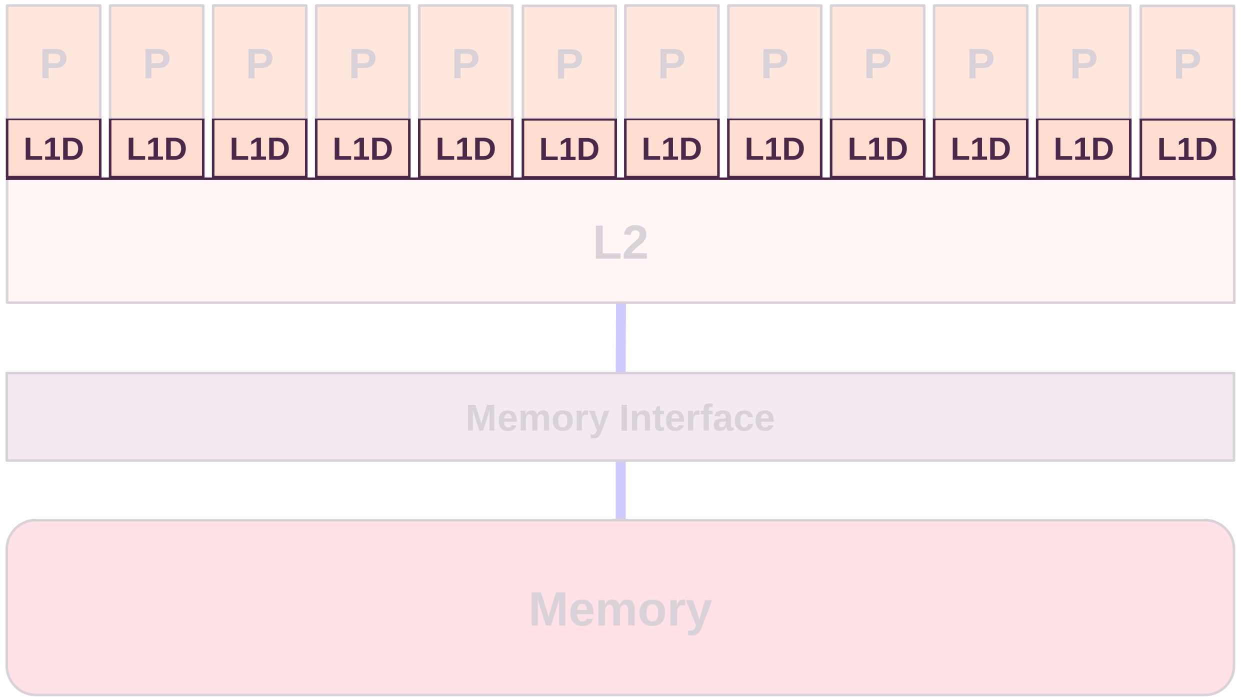
Core

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

For example on GCC compiler use :

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

A64FX – FX700

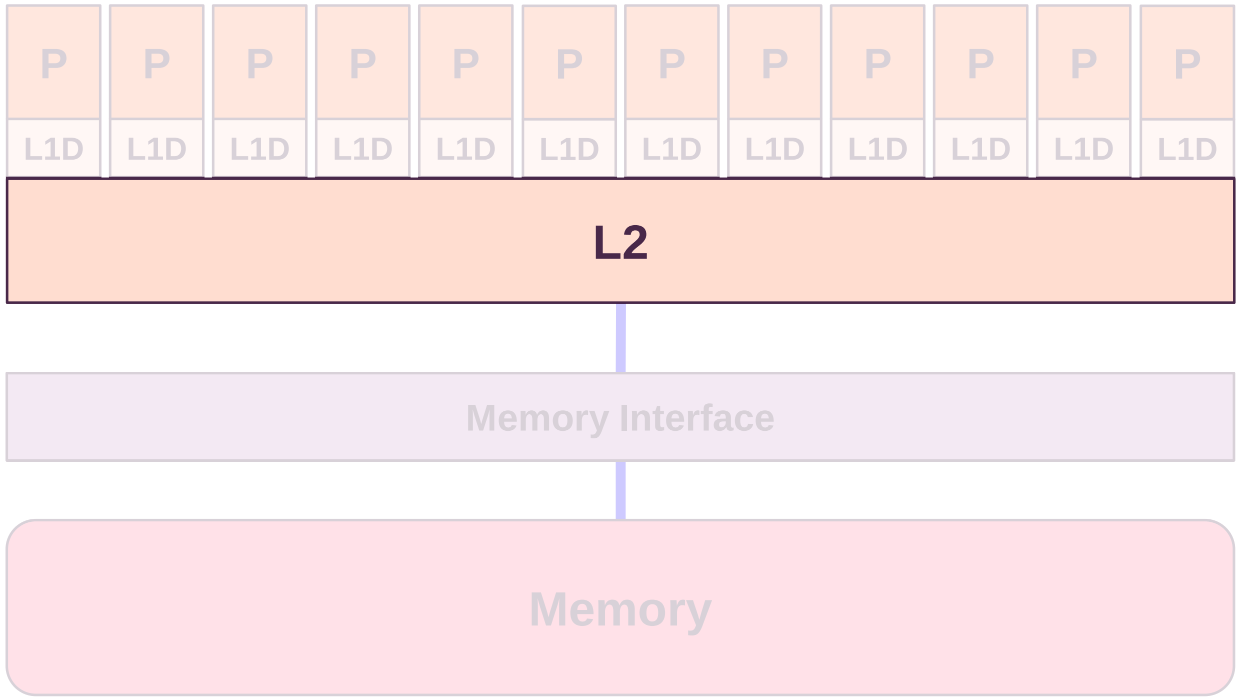


1 CMG

L1D cache

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

A64FX – FX700

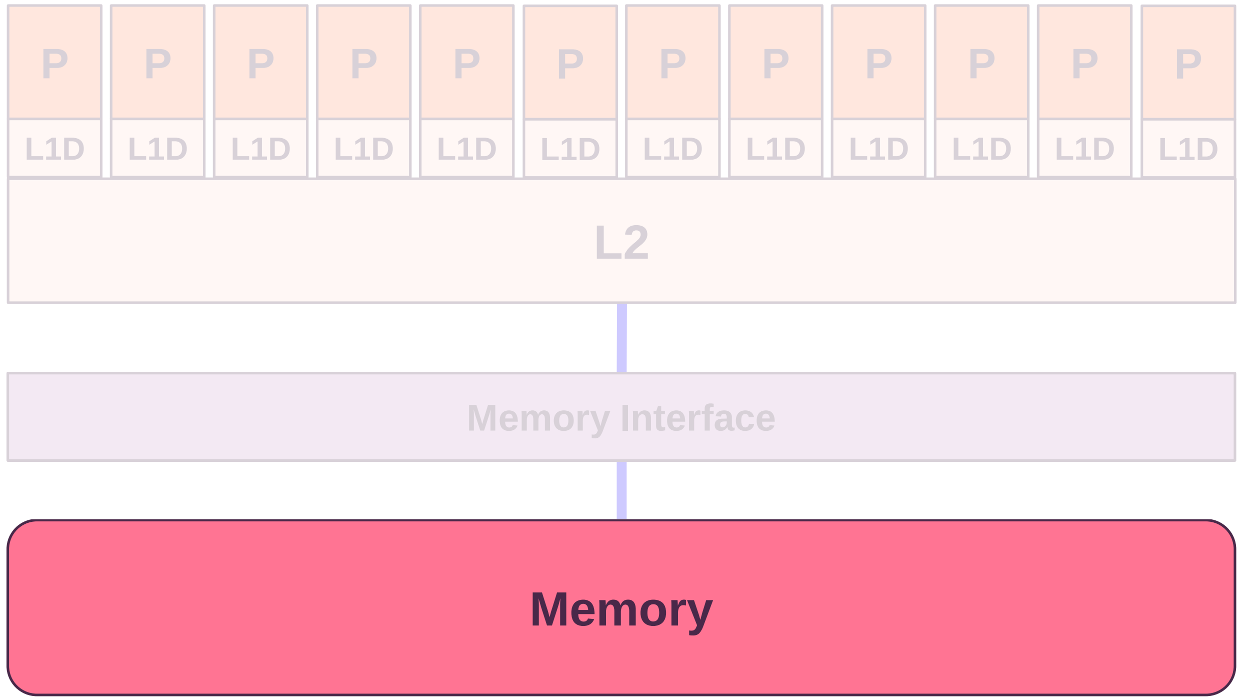


1 CMG

L2 cache

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

A64FX – FX700

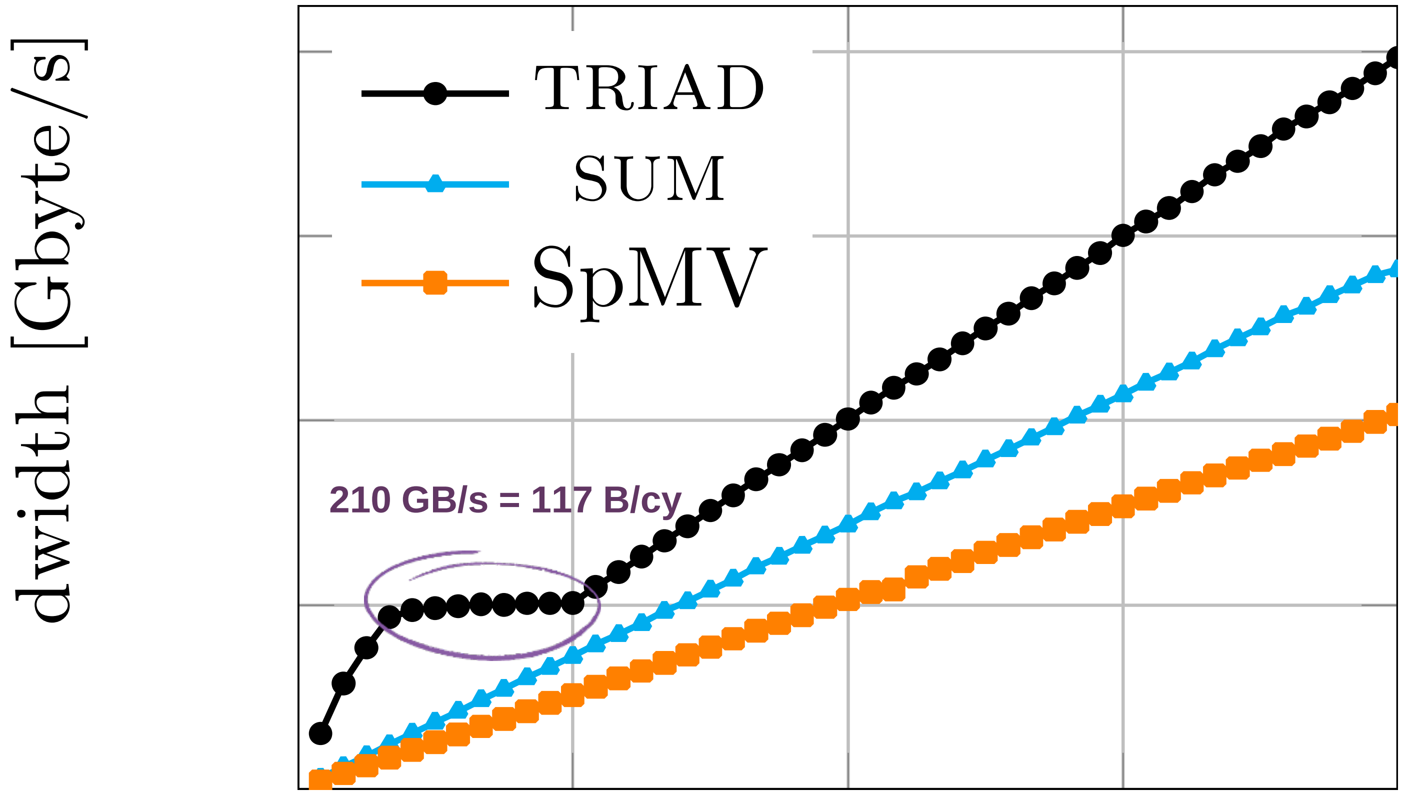


1 CMG

Main memory

- Size : 4 x 8 GiB
- Type : HBM2

Motivation

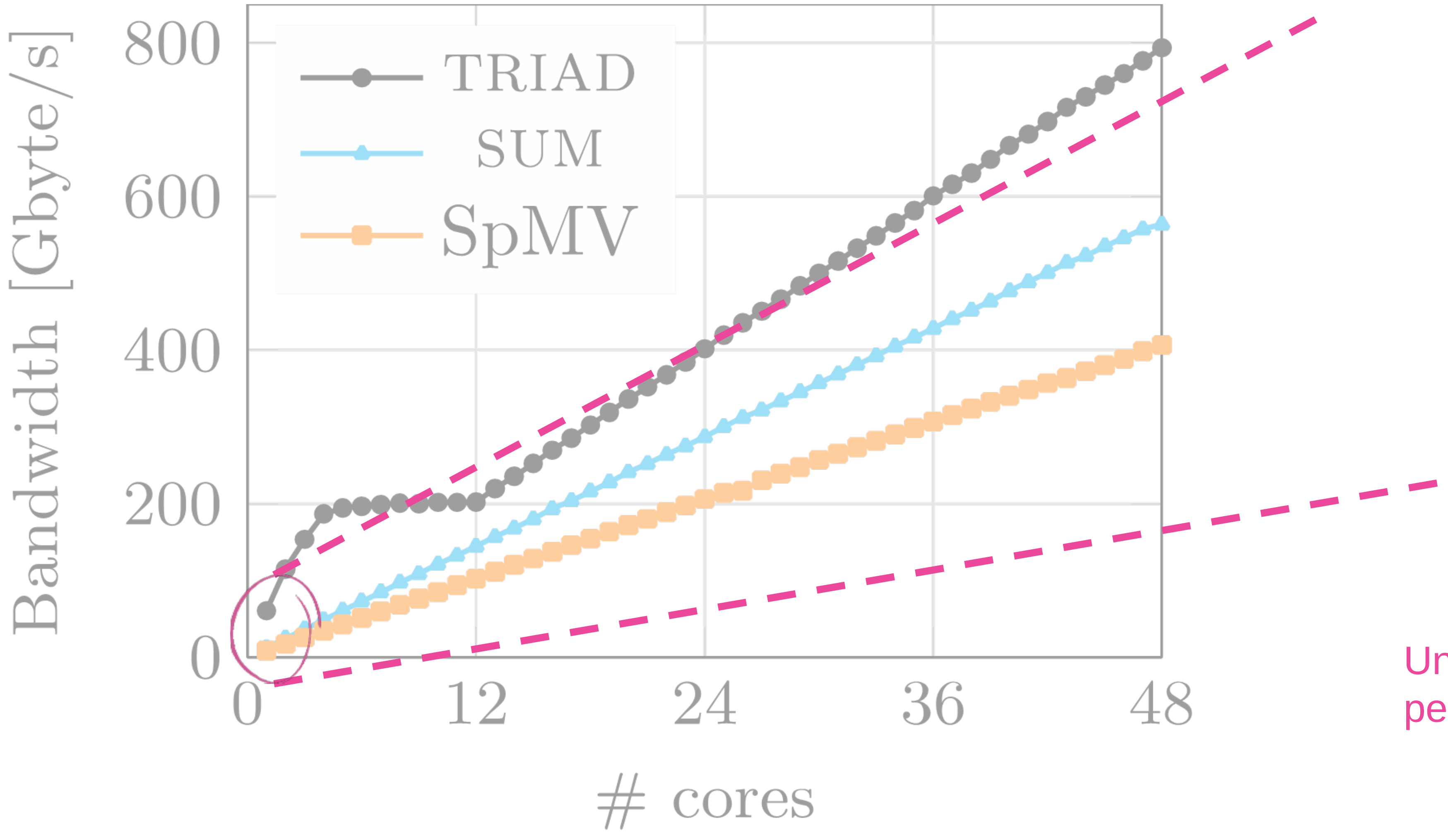


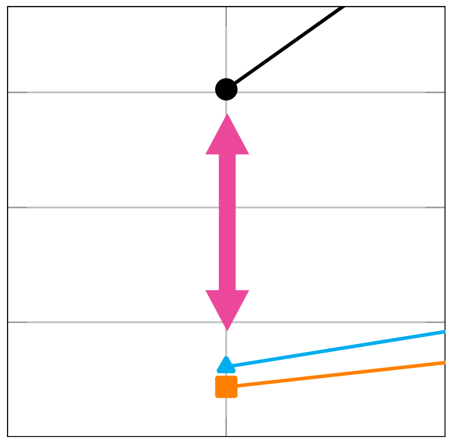
Thread pinning : Compact

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 = Ax`)?

Motivation

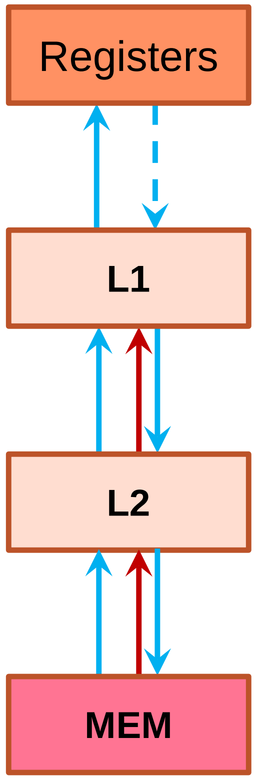




Understanding single-core performance is the key !

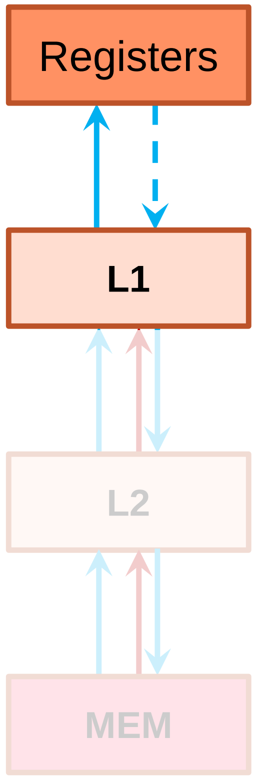
Thread pinning : Compact

Motivation → ECM model



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

Motivation → ECM model

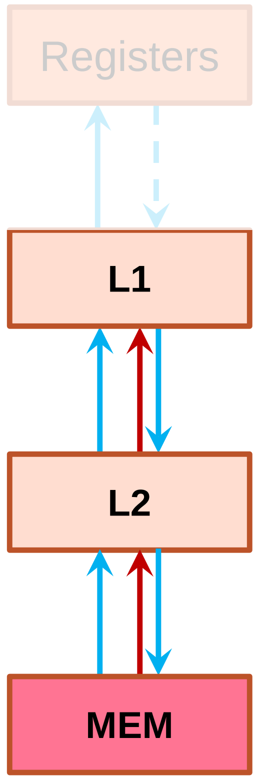


Execution-Cache-Memory (ECM) model helps us to understand and analyze the single-core performance.

3 major components :

1) In-core

Motivation → ECM model



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

3 major components :

1) In-core

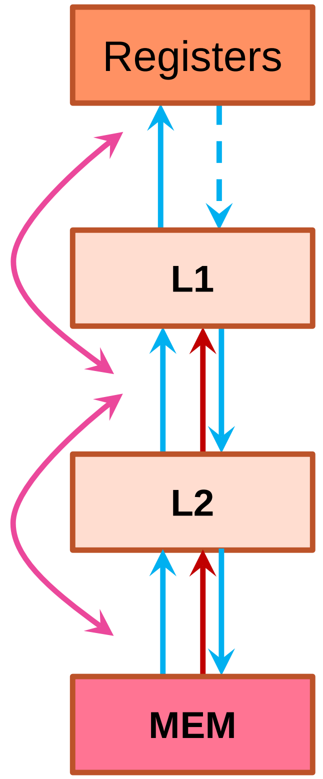
2) Data transfer through memory hierarchies

SC20

Motivation → ECM model

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Universität Regensburg



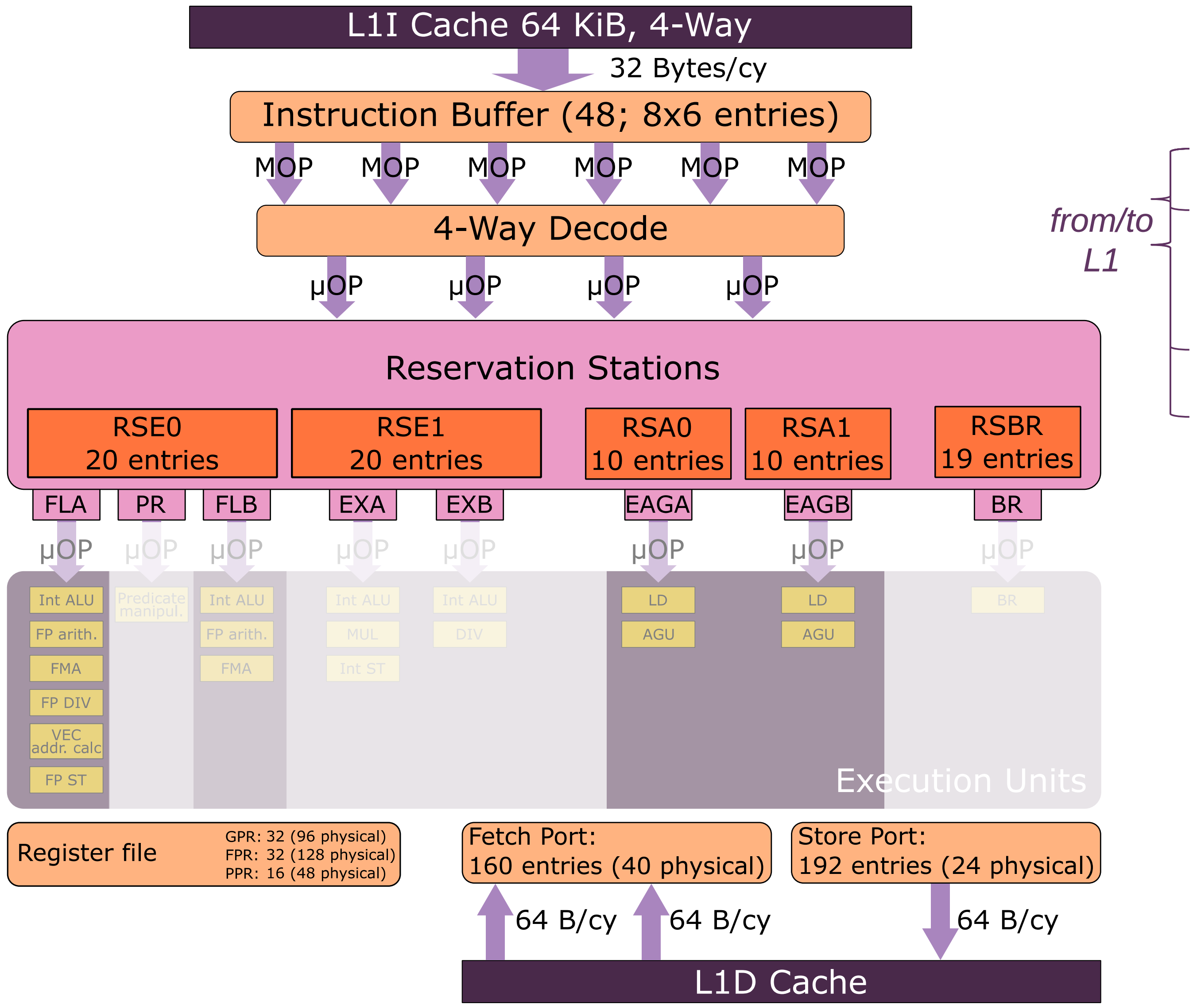
Can these transfers be overlapped or not ?

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

3 major components :

1) In-core
2) Data transfer through memory hierarchies
3) Overlap hypothesis

ECM model → In-core



[1]horizontal recursive add

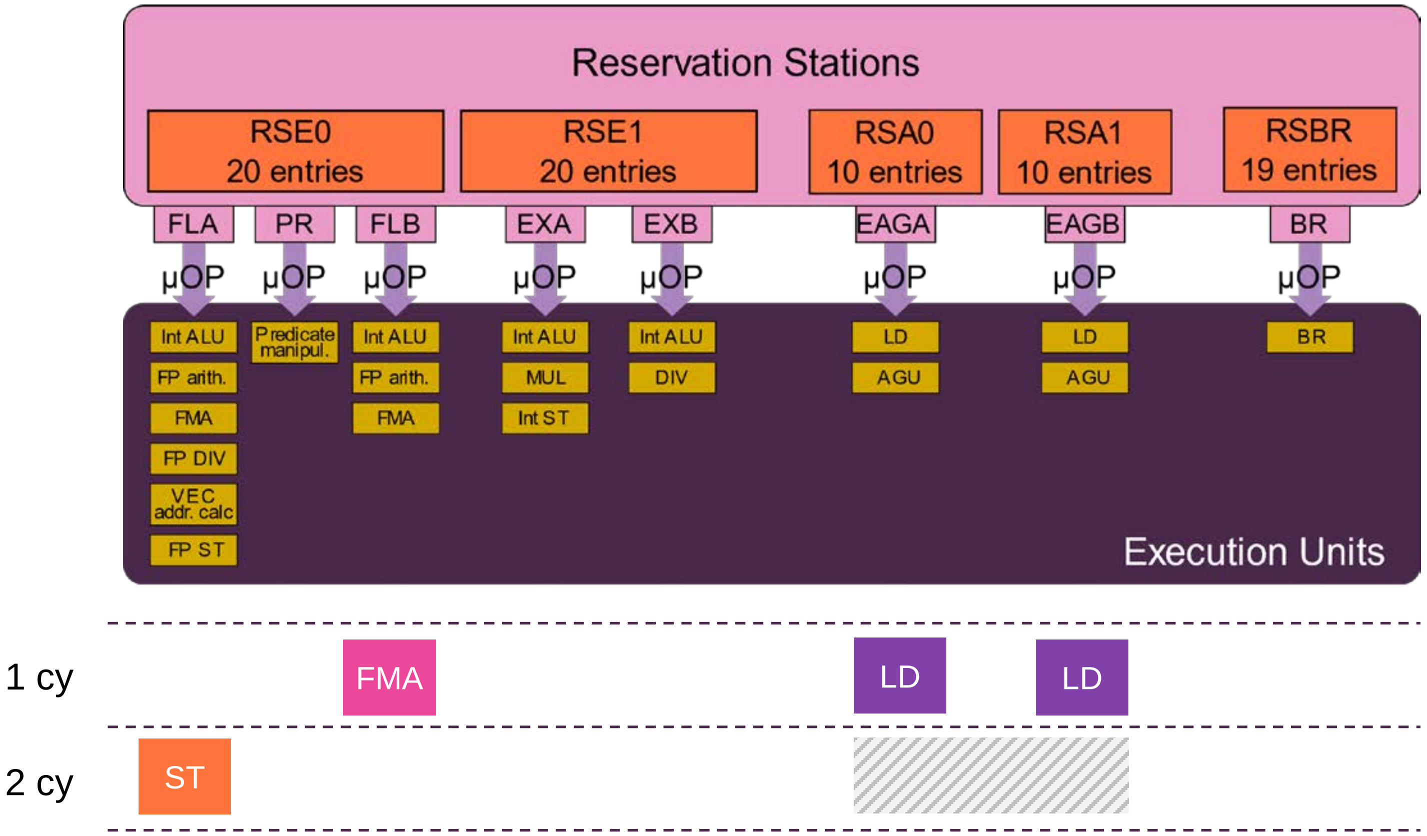
ECM model → In-core

STREAM TRIAD

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

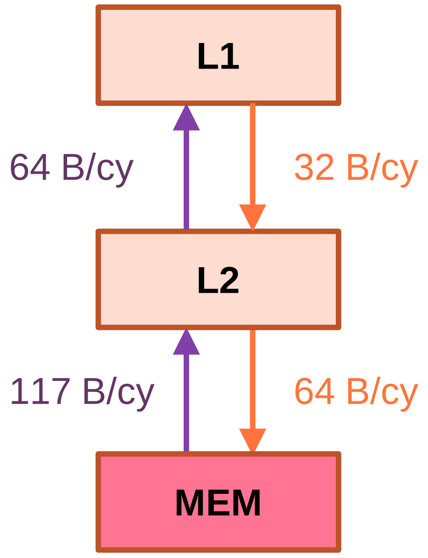
```
.L18:
  ld1d z4.d, p5/z, [x21, x9, lsl 3]
  ld1d z5.d, p5/z, [x20, x9, lsl 3]
  fmad z5.d, p5/m, z2.d, z4.d
  st1d z5.d, p5, [x19, x9, lsl 3]
  add x8, x9, 8
  whilelo p5.d, w8, w7
  b.any .L18
```

2cy / VL



ECM model → Memory hierarchy

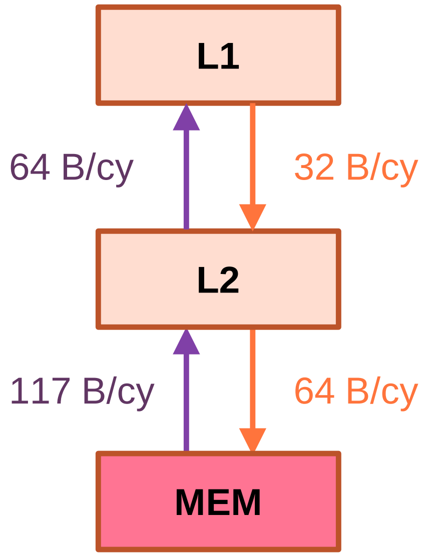
Machine model
FX700



ECM model → Memory hierarchy

Machine model

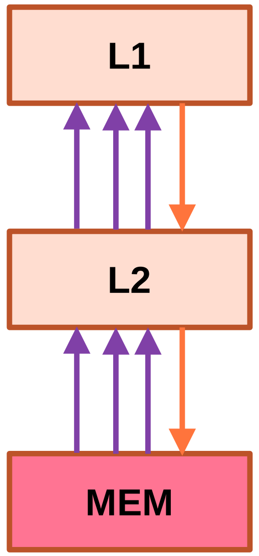
FX700



Application model

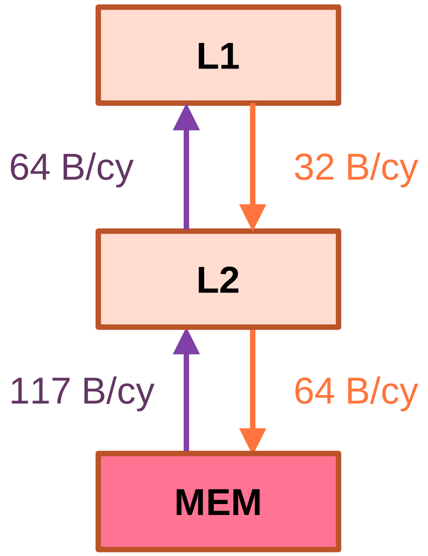
TRIAD

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

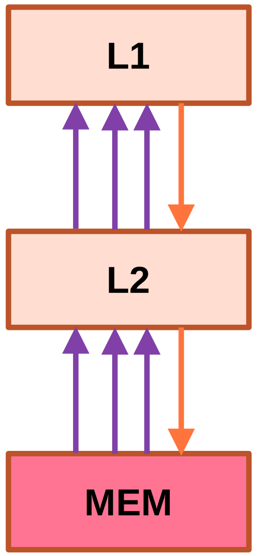


ECM model → Memory hierarchy

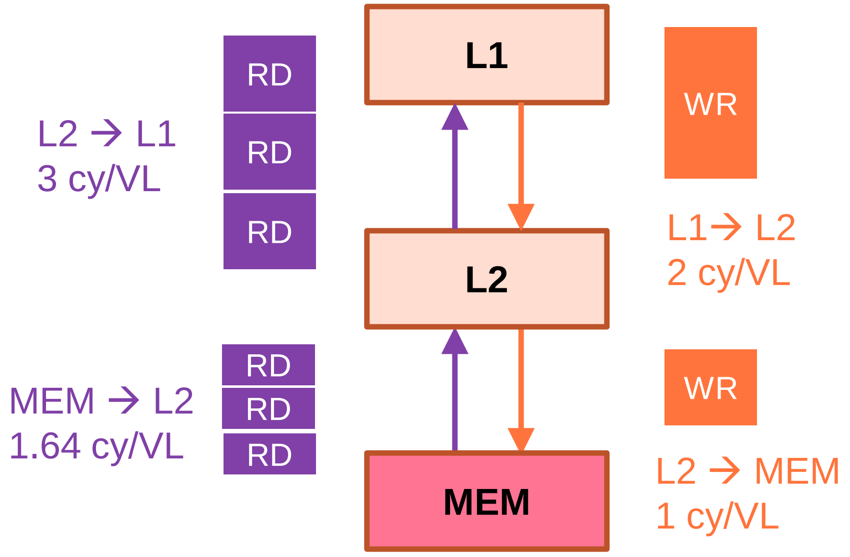
Machine model
FX700



Application model
TRIAD
`a[i] = b[i] + s*c[i]`

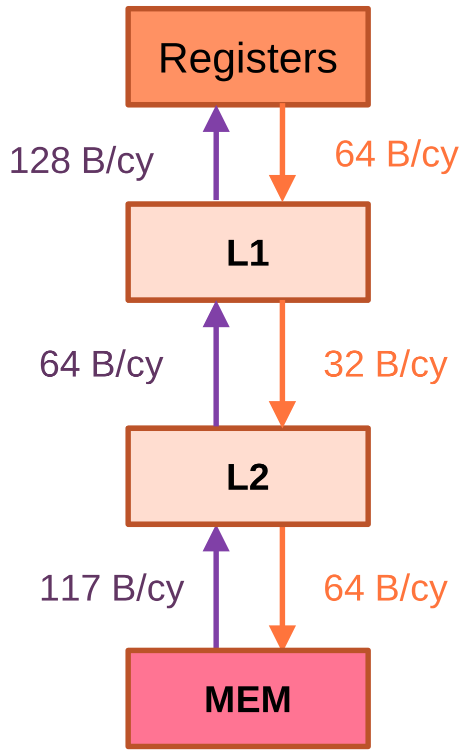


ECM prediction
TRIAD on FX700

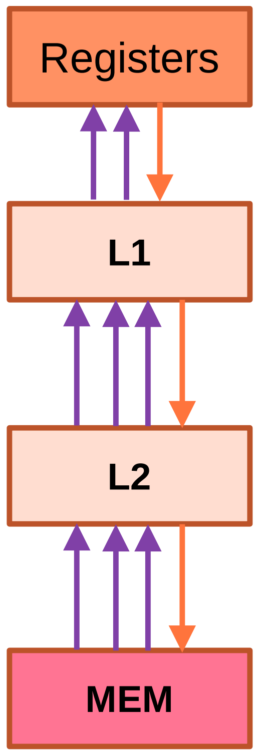


ECM model → Memory hierarchy + In-core

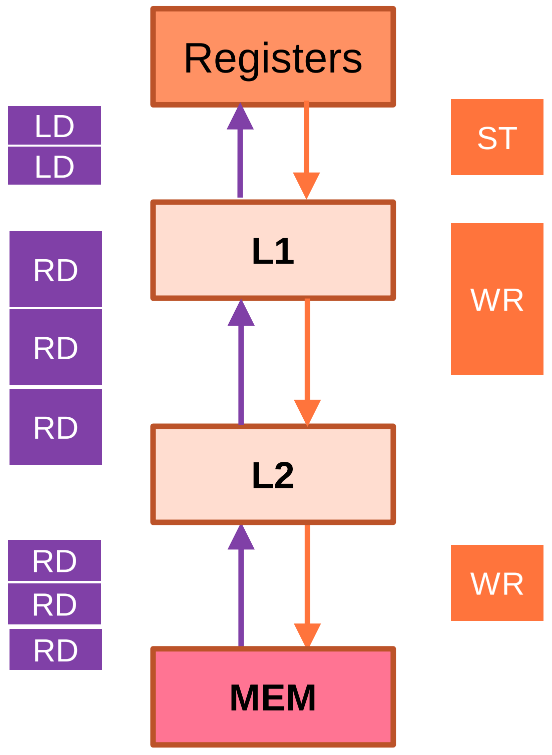
Machine model
FX700



Application model
TRIAD
`a[i] = b[i] + s*c[i]`



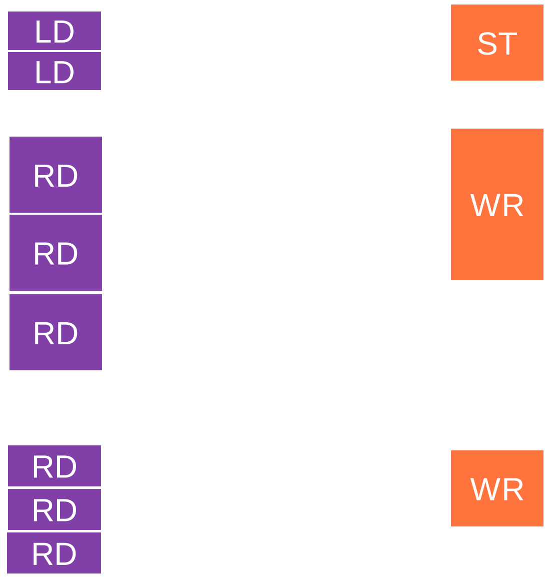
ECM prediction
TRIAD on FX700



ECM model → Overlap hypothesis

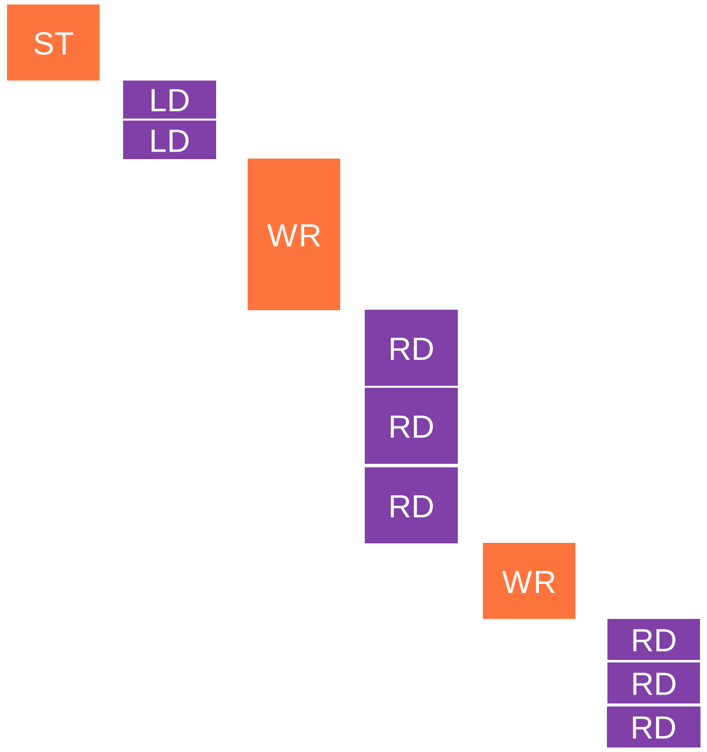
How do these boxes overlap?

ECM prediction
TRIAD on FX700



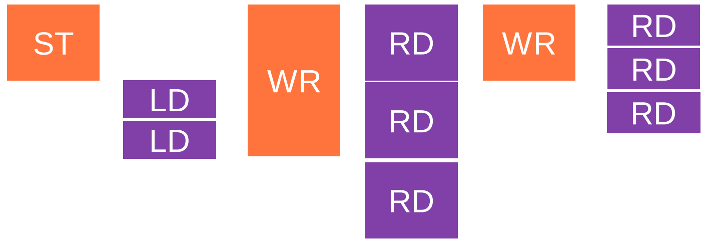
ECM model → Overlap hypothesis

Hypothesis 1 : No overlap



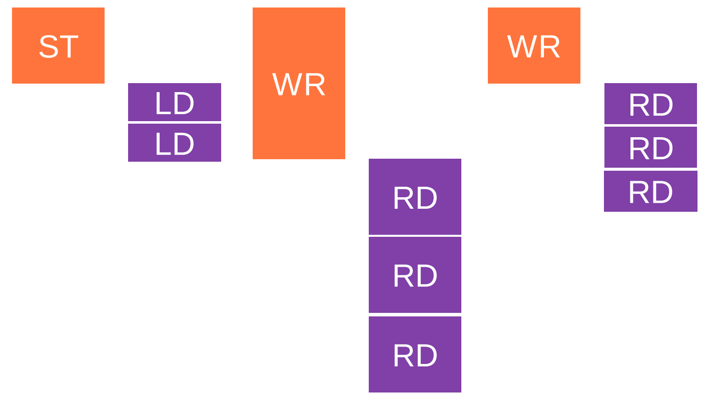
ECM model → Overlap hypothesis

Hypothesis 2 : Full overlap



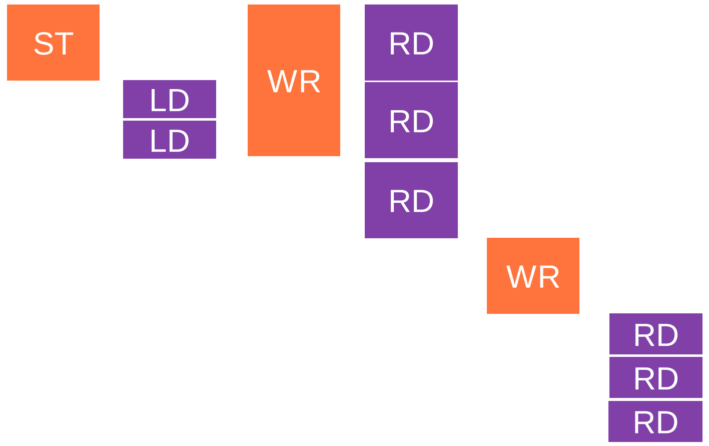
ECM model → Overlap hypothesis

Hypothesis 3 : Full overlap + half-duplex



ECM model → Overlap hypothesis

Hypothesis 4 : L1L2 overlap + half-duplex at MEM



There are numerous combinations.

How do we find the correct one?

Compare measurements
with predictions.

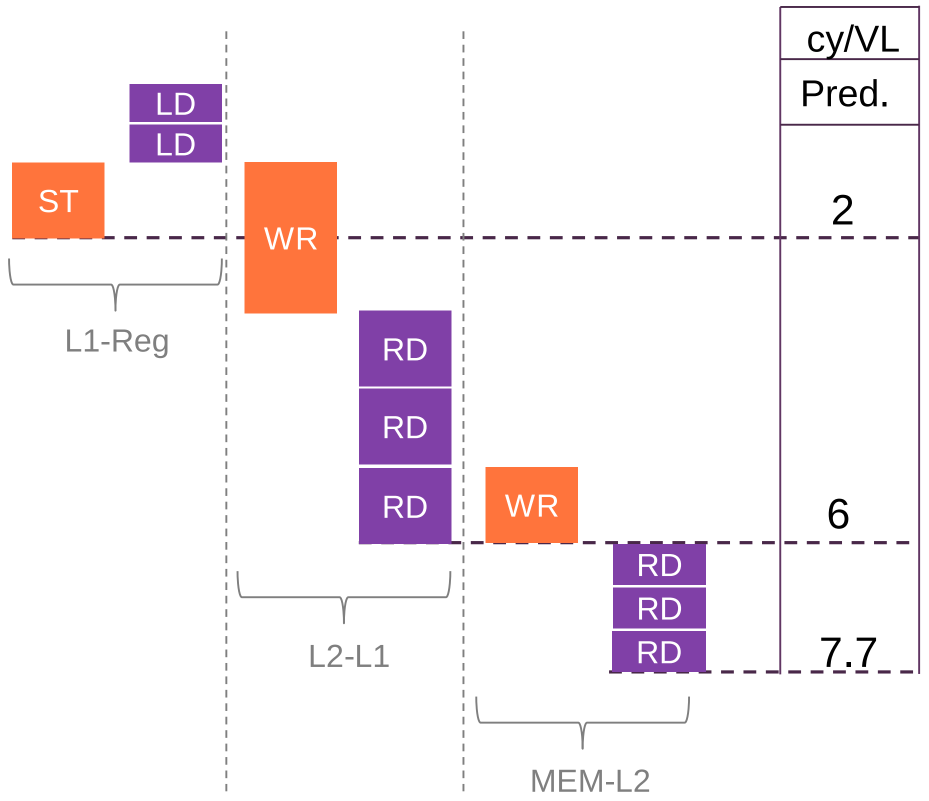
ECM model → Overlap hypothesis

There are numerous combinations.

How do we find the correct one?

Compare measurements
with predictions.

The best hypothesis for FX700



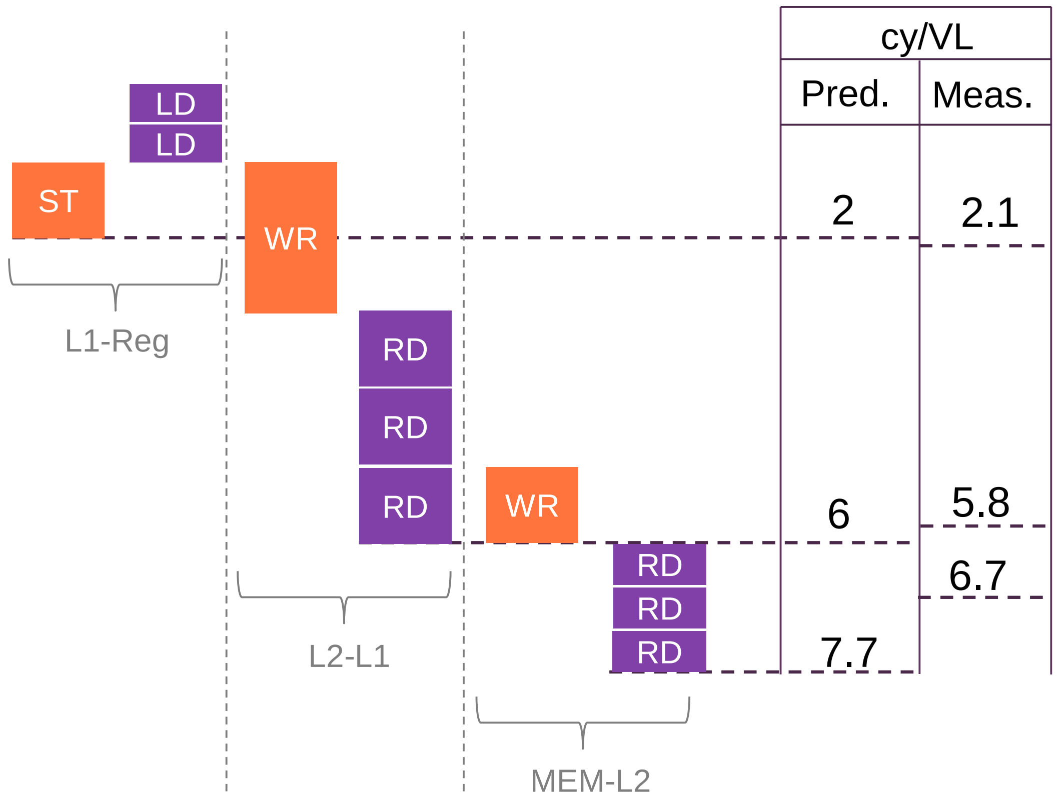
ECM model → Overlap hypothesis

There are numerous combinations.

How do we find the correct one?

Compare measurements with predictions.

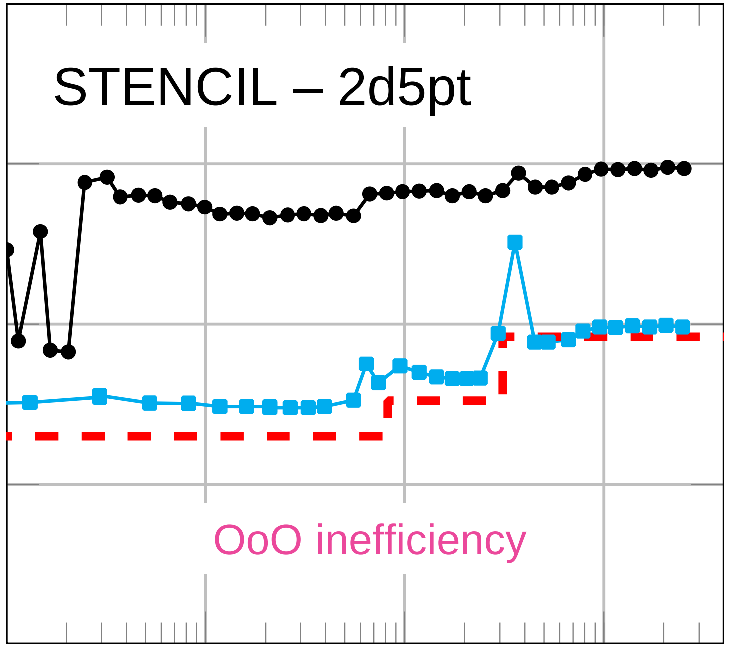
The best hypothesis for FX700

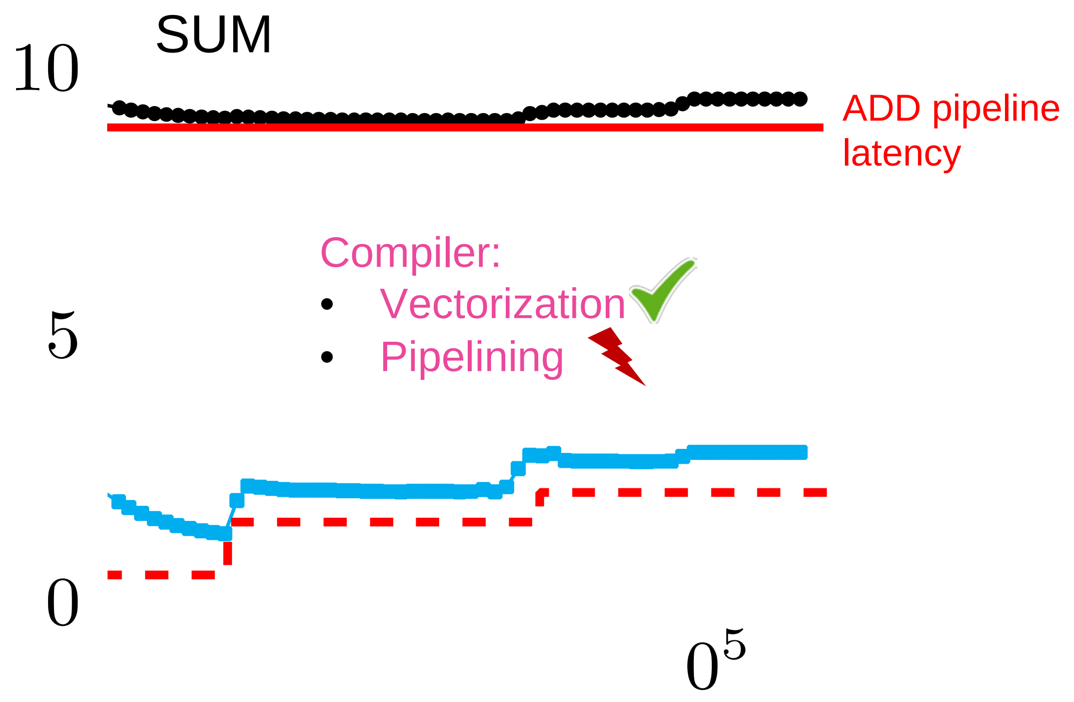


A systematic way of identifying overlap hypothesis is presented in : Hofmann et.al., 2020, Bridging The Architecture Gap: Abstracting Performance-relevant Properties Of Modern Server Processors, https://doi.org/10.14529/jsfi200204

ECM model → Insights

Unrolling plays an important role

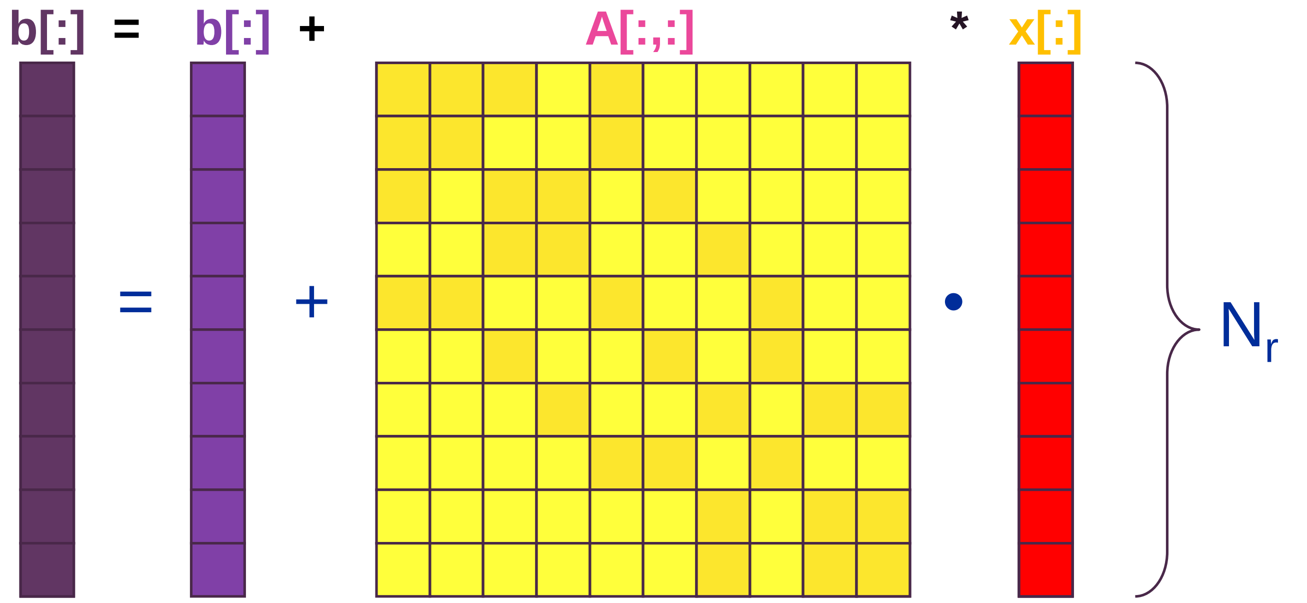




■ Unrolling factor=1 ■ Unrolling factor=8 🟥 ECM prediction

SpMV

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



General case: some indirect addressing required!

In Compressed Row Storage (CRS) format

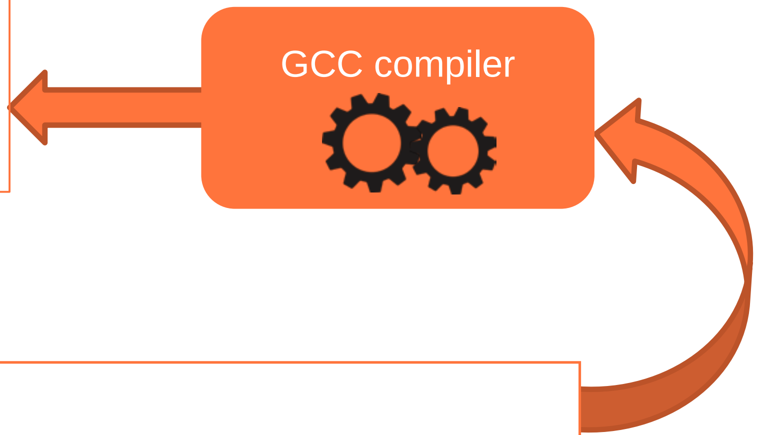
```
for i = 0:nrows-1 //Long outer-loop
    for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner-loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

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

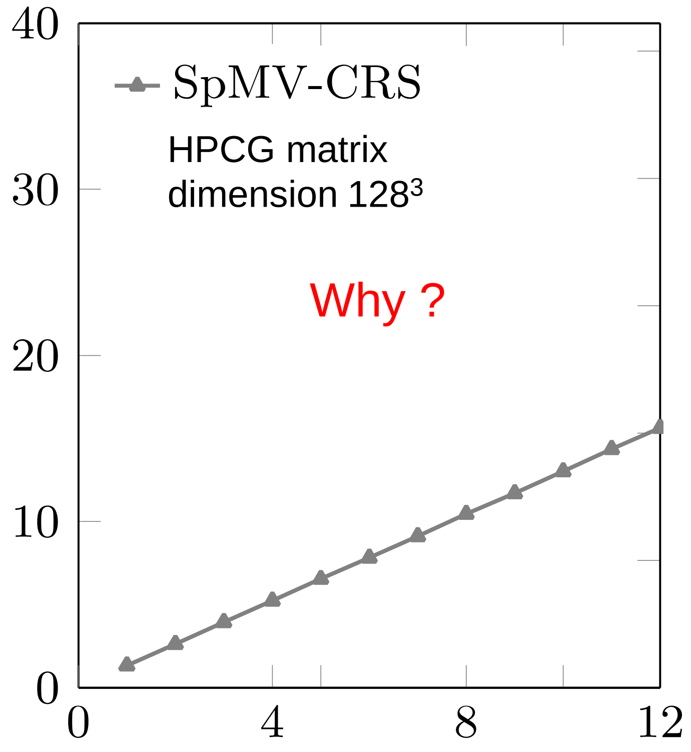
```
for i = 0:nrows-1 //Long outer-loop
    for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner-loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

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
```

FMA3: Update **z1.d**

Latency: 9 cycles

Loop length : 27
HPCG matrix

Horizontal add of 512-bit register

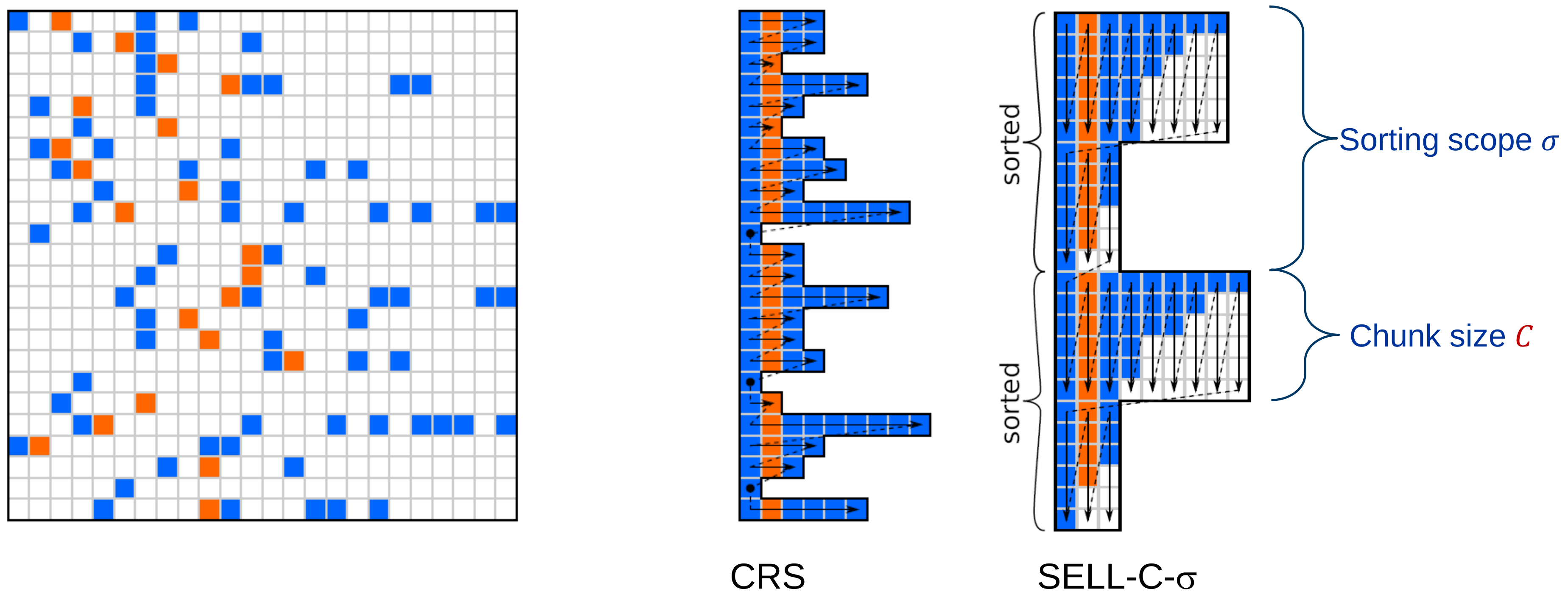
Throughput = 11.5 cycles

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 -> SELL-C-σ

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

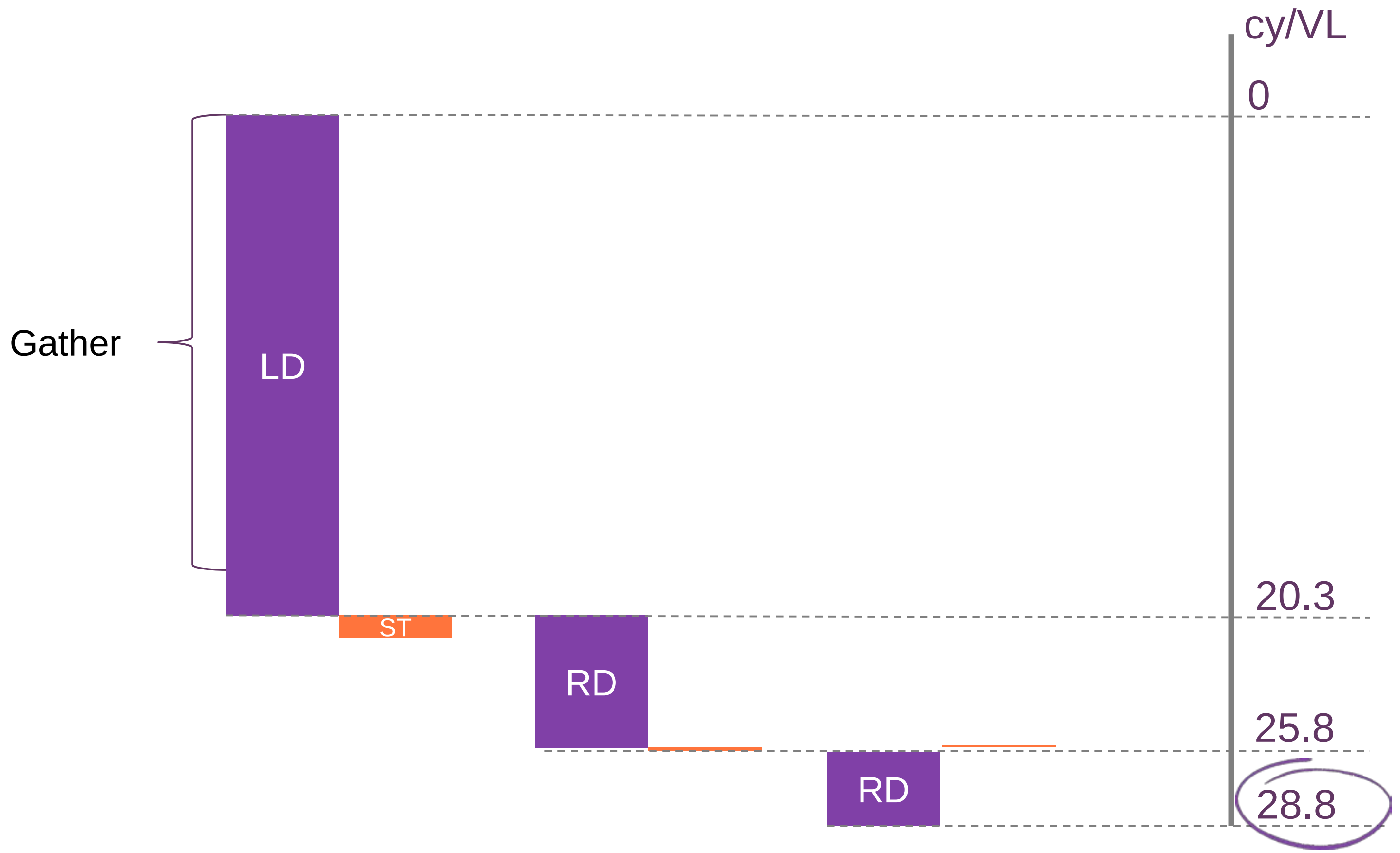


Benefits:

- 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

SpMV → SELL-C-σ → ECM

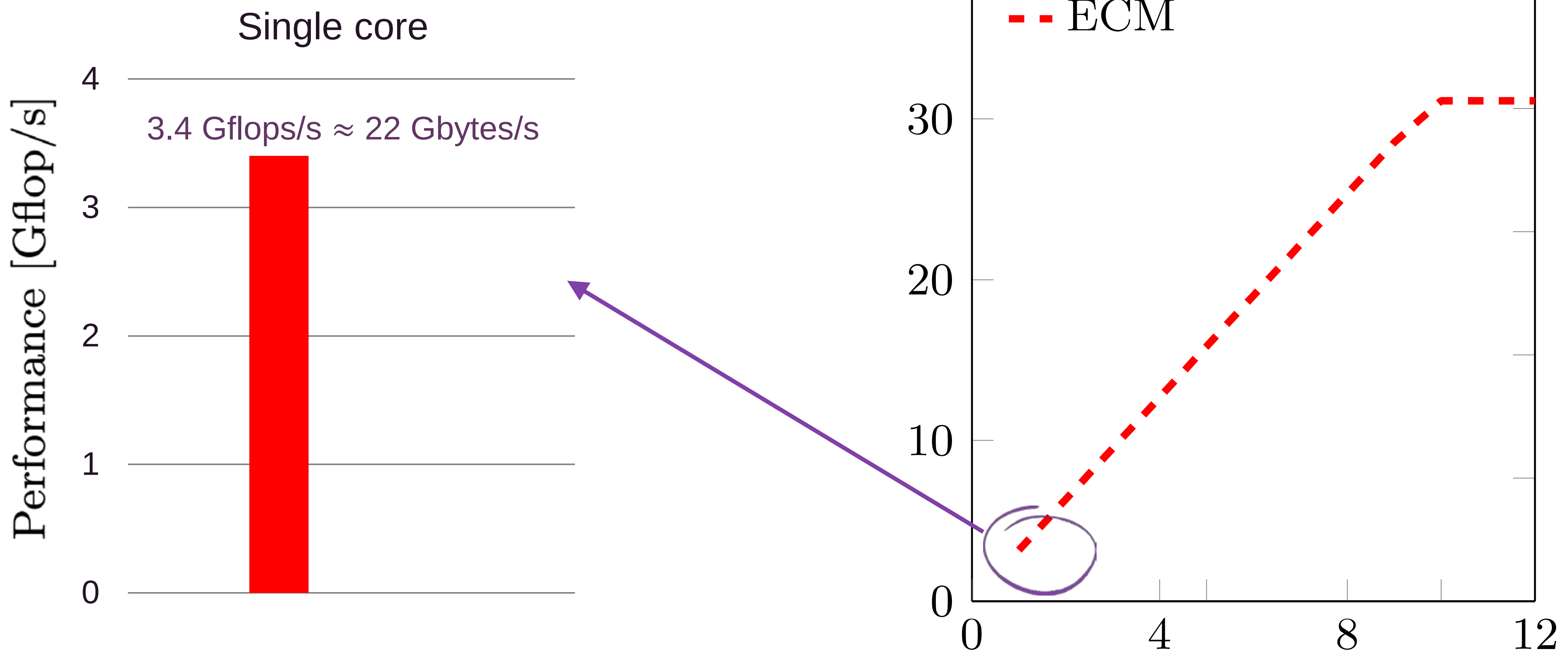


= 3.4 Gflops/s ≈ 22 Gbytes/s

SC20

SpMV → SELL-C-σ → ECM

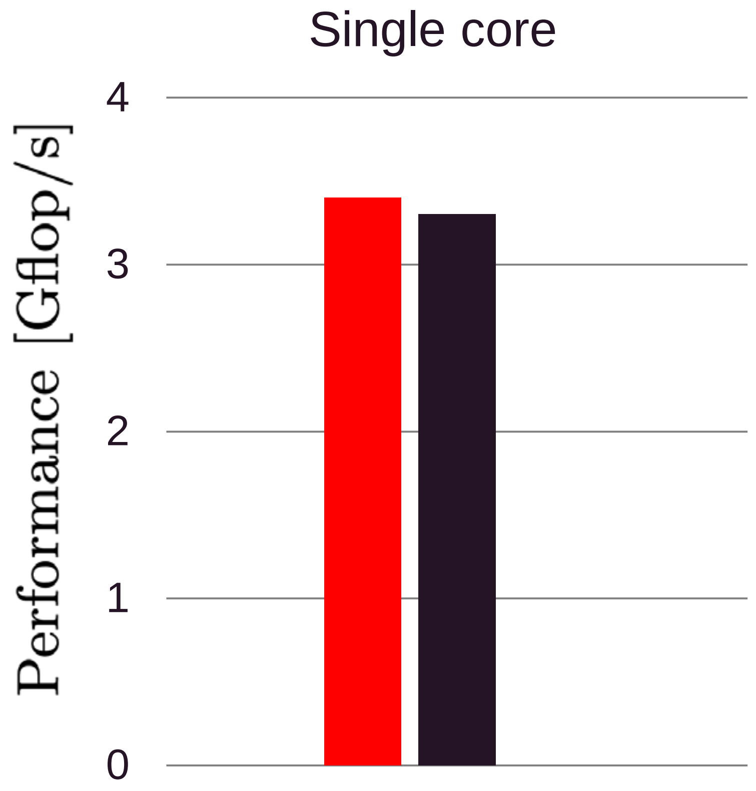
Can we saturate now ?

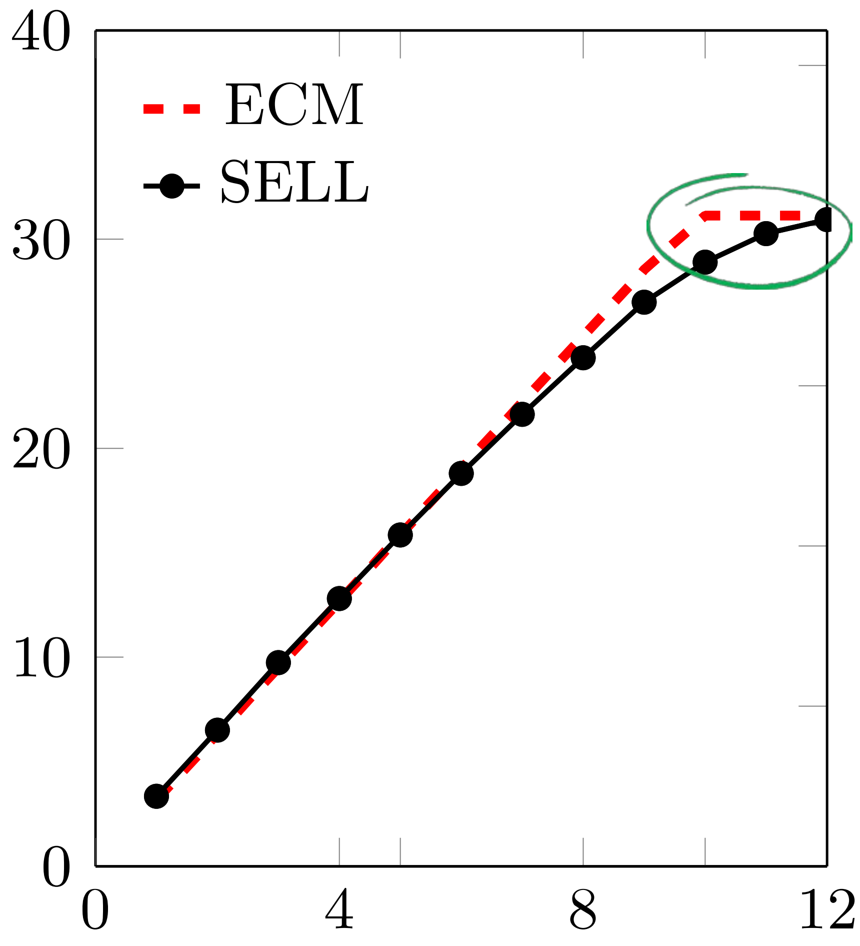


HPCG matrix, dimension 128^3

SpMV → SELL-C-σ → ECM

Can we saturate now ? Yes, but needs almost all cores

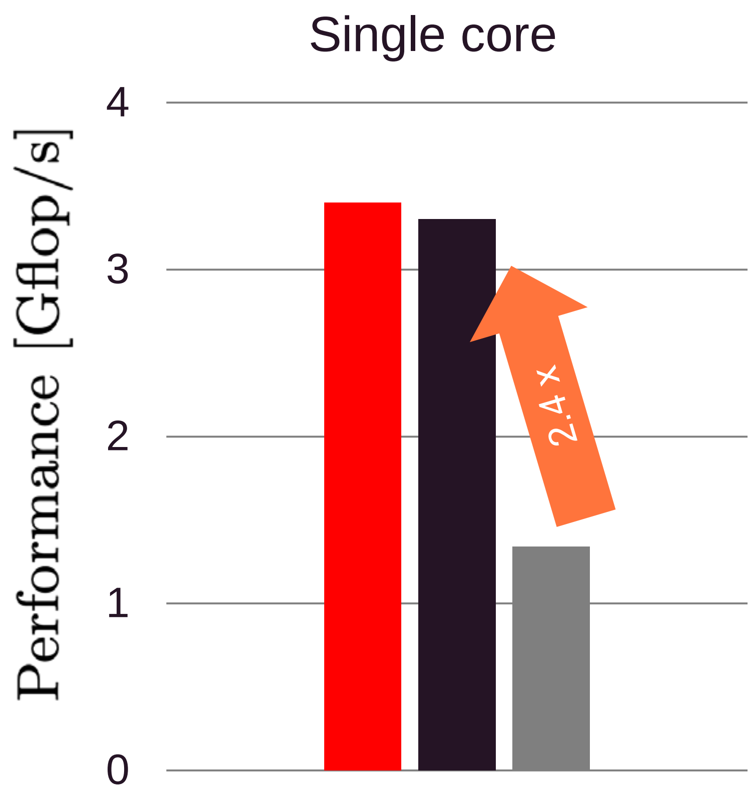


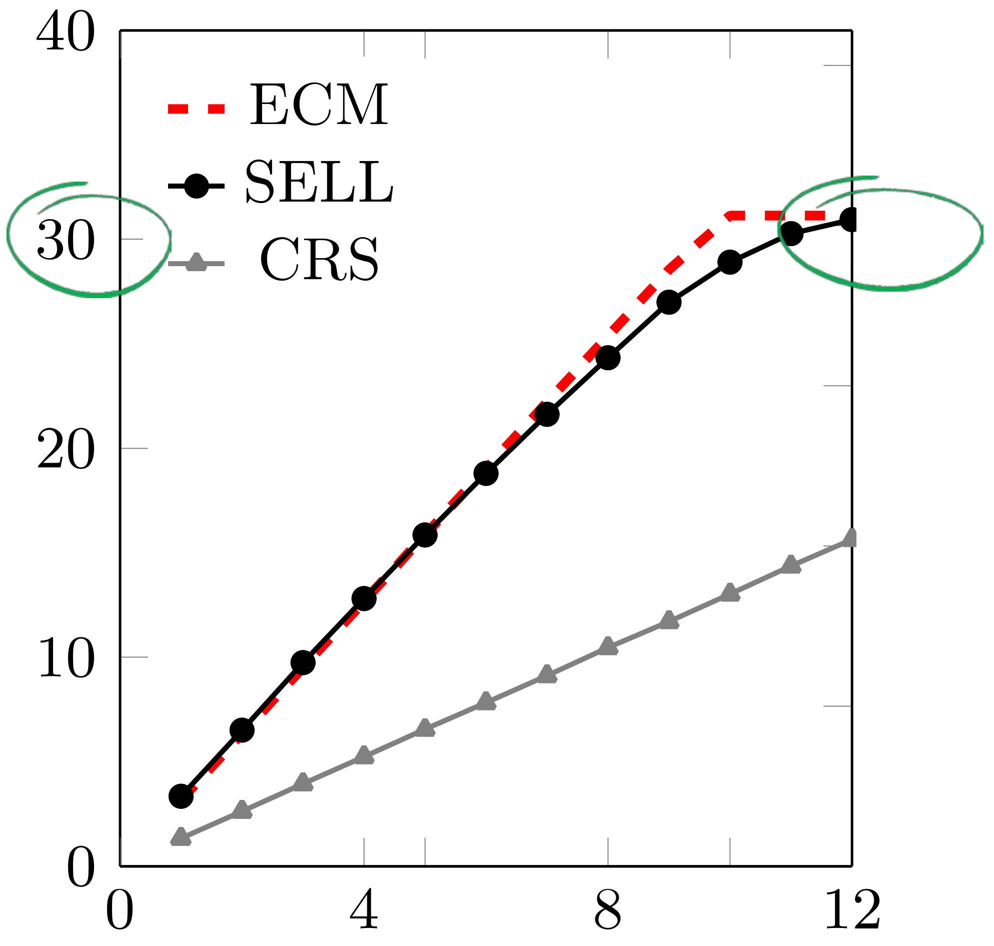


HPCG matrix, dimension 128^3

SpMV → SELL-C-σ → ECM

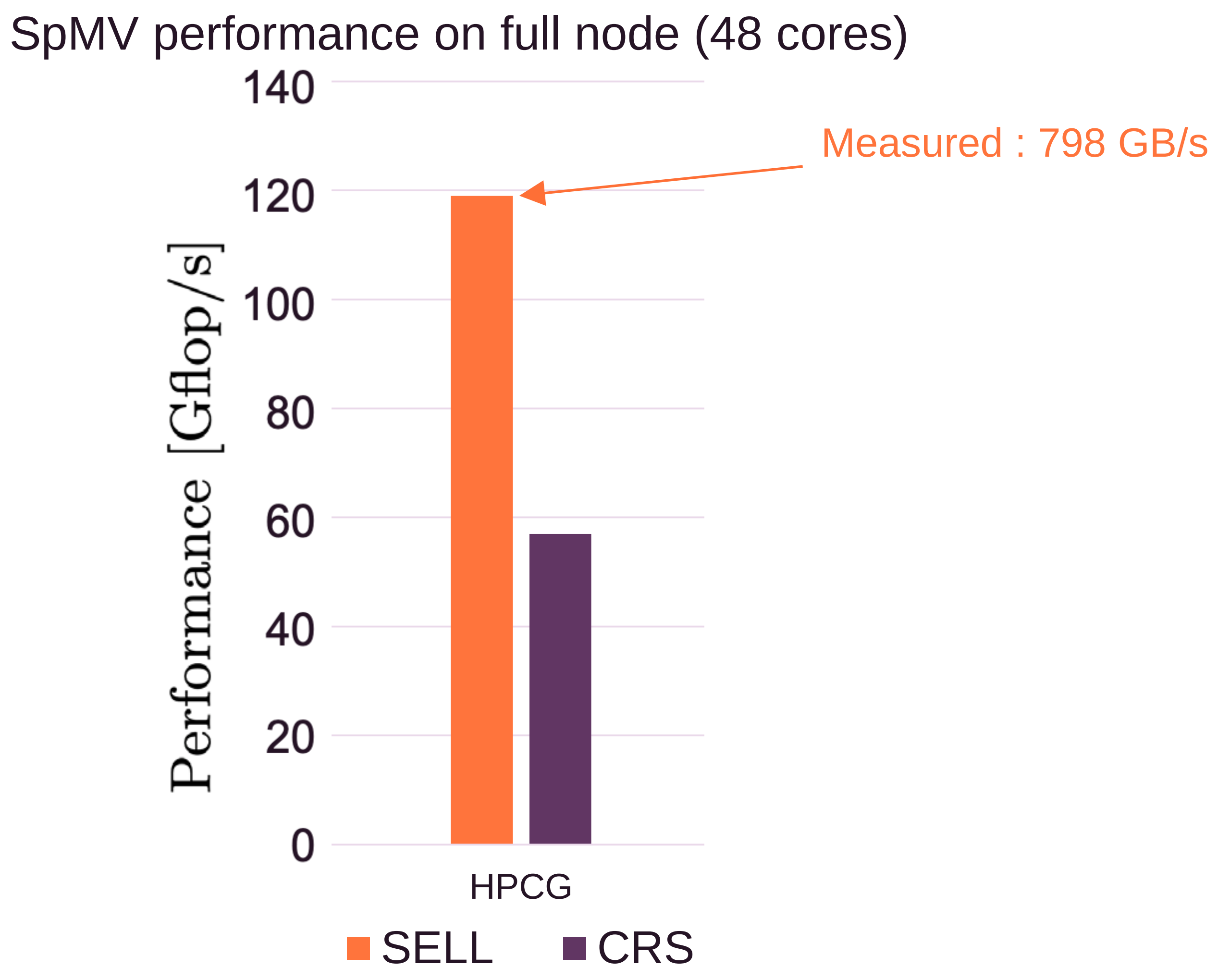
Can we saturate now ?





HPCG matrix, dimension 128^3

SpMV



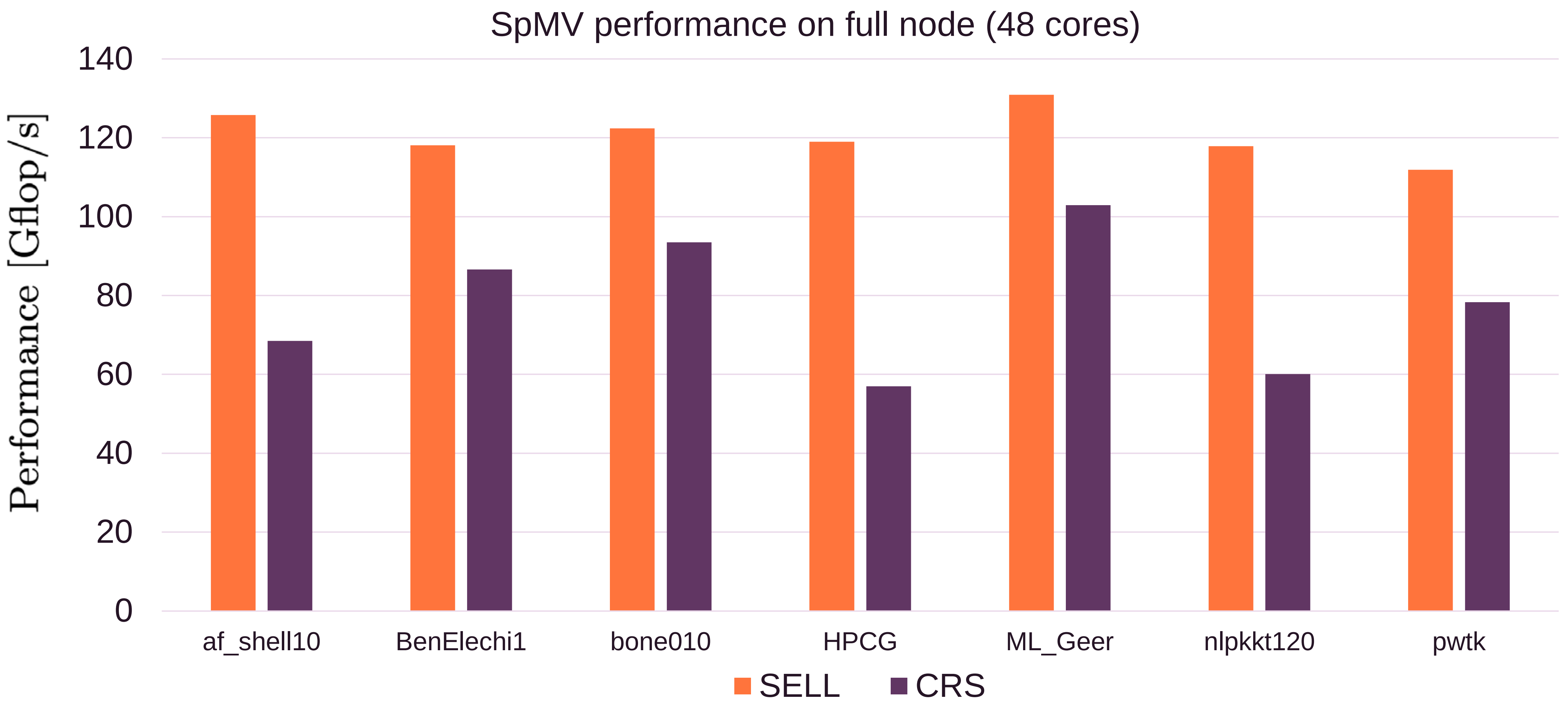
SpMV performance on full node (48 cores)
Measured : 798 GB/s
Performance [Gflop/s]
0
20
40
60
80
100
120
140
HPCG
SELL
CRS

SC20

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Universität Regensburg

SpMV



Matrices from SuiteSparse Matrix Collection : https://suitesparse-collection-website.herokuapp.com

Conclusion

- High single core performance is crucial.
- ECM model was established and utilized to analyze the single core performance.
- The partial overlapping memory hierarchy allows for high single-core memory bandwidth.
- Proper single core optimizations have to be done to hide long floating point latency and inefficiencies in OoO.
- For SpMV we were able to saturate the bandwidth with SELL-C-σ format.

Thank you

Questions ?

KONWIHR

SFB TRR55