ERLANGEN REGIONAL
COMPUTING CENTER

Insight into stencil performance by analytic modeling

Georg Hager
Erlangen Regional Computing Center (RRZE)

Seminar on Advanced Stencil Code Engineering
April 13-17, 2015, Schloss Dagstuhl, Wadern, Germany

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

References

- J. Treibig and G. Hager: *Introducing a Performance Model for Bandwidth-Limited Loop Kernels*. Proceedings of the Workshop “Memory issues on Multi- and Manycore Platforms” at PPAM 2009, the 8th International Conference on Parallel Processing and Applied Mathematics, Wroclaw, Poland, September 13-16, 2009. Lecture Notes in Computer Science Volume 6067, 2010, pp 615-624.
  DOI: 10.1007/978-3-642-14390-8_64 (2010).
- G. Hager, J. Treibig, J. Habich, and G. Wellein: *Exploring performance and power properties of modern multicore chips via simple machine models*.
  Concurrency and Computation: Practice and Experience,
  DOI: 10.1002/cpe.3180 (2013).
- M. Wittmann, G. Hager, T. Zeiser, J. Treibig, and G. Wellein: *Chip-level and multi-node analysis of energy-optimized lattice-Boltzmann CFD simulations.*
  Accepted for publication in Concurrency Computat.: Pract. Exper. (2015)
  Preprint: arXiv:1304.7664
- H. Stengel, J. Treibig, G. Hager, and G. Wellein: *Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model*. Accepted for ICS’15, the 29th International Conference on Supercomputing, Newport Beach, CA, June 8-11, 2015.
  DOI: 10.1145/2751205.2751240, Preprint: arXiv:1410.5010

Further references

- M. Wittmann, G. Hager, J. Treibig and G. Wellein: *Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters.* Parallel Processing Letters **20** (4), 359-376 (2010).
DOI: 10.1142/S0129626410000296

- J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein: *Pushing the limits for medical image reconstruction on recent standard multicore processors.* International Journal of High Performance Computing Applications **27**(2), 162-177 (2013).
DOI: 10.1177/1094342012442424

- S. Kronawitter, H. Stengel, G. Hager, and C. Lengauer: *Domain-Specific Optimization of Two Jacobi Smoother Kernels and Their Evaluation in the ECM Performance Model*. Parallel Processing Letters **24**, 1441004 (2014).
DOI: 10.1142/S0129626414410047

Motivation

- There are so many
  - potential stencil structures (2D/3D, long-/short-range,…),
  - text book optimizations (register / spatial / temporal blocking,…),
  - parameters for optimization (blocking / unrolling factor,…),
  - parameters for execution (OpenMP schedule, clock speed, #cores).

- Basic questions addressed by analytic performance models
  - What is the bottleneck of my stencil implementation?
    → Choose appropriate optimization technique
  - What is the next bottleneck I will hit and “how far” is it from the first?
    → This yields the performance potential of the optimization
  - Impact of processor frequency and socket scalability
    → Choose appropriate execution parameters
    → Energy-optimized operating point

The “classic” Roofline Model[1,2]

1. ***P***P_{max} = Applicable peak performance of a loop
   (this is not necessarily P_{peak})

2. ***I*** = Operational intensity (“work” / byte transferred) over the slowest data path utilized (“the bottleneck”)

3. b_S = Applicable peak bandwidth of the slowest data path utilized (hardware feature)

Expected performance:

$$P = \min(P_{max}, I \cdot b_S)$$

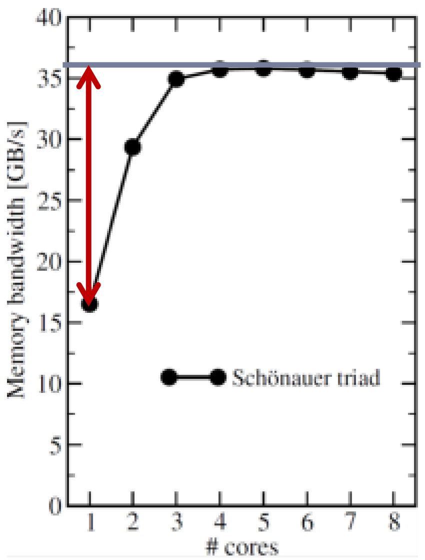
[F/B]

[B/s]

[1] W. Schönauer: Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers. (2000)

[2] S. Williams: Auto-tuning Performance on Multicore Computers. UCB Technical Report No. UCB/EECS-2008-164. PhD thesis (2008)

Agenda



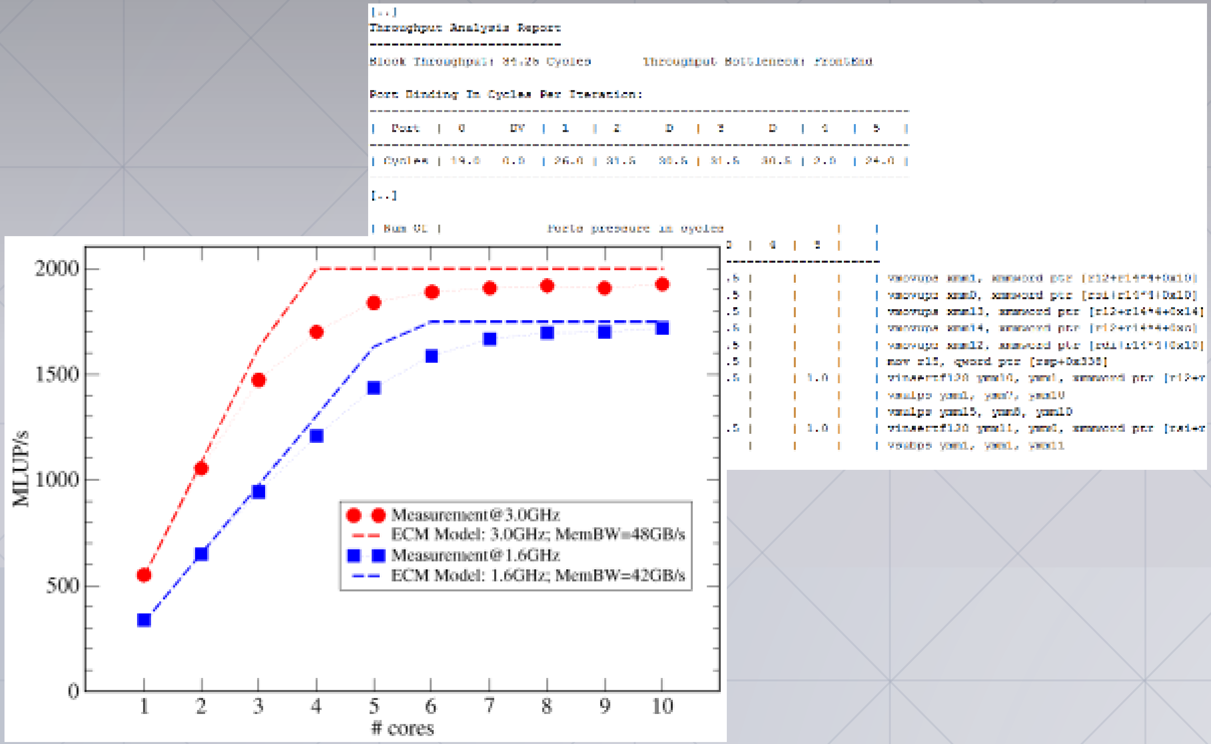
- ECM model
  - Basic rules, non-overlap
  - Notation
  - Saturation and comparison with Roofline
  - Example: array sum

- Case study 1: 5-pt 2D stencil (“Jacobi”)
- Case study 2: UXX stencil (SP/DP)
- Case study 3: 25-pt long-range stencil (SP)

- First steps towards automated model construction: the “kerncraft” tool

- Summary

THE ECM MODEL

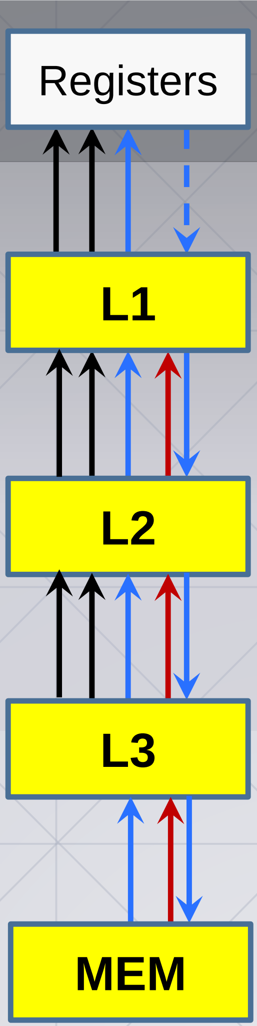


```
[..]
Throughput Analysis Report
--------------------------
Block Throughput: 34.25 Cycles       Throughput Bottleneck: FrontEnd

Port Binding In Cycles Per Iteration:
---------------------------------------------------------------------------------------
|  Port  |  0   -  DV  |  1   |  2   -  D   |  3   -  D   |  4   |  5   |
---------------------------------------------------------------------------------------
| Cycles | 19.0    0.0 | 26.0 | 31.5   30.5 | 31.5   30.5 | 2.0  | 24.0 |
---------------------------------------------------------------------------------------

[..]

| Num Of |                Ports pressure in cycles                |    |
                                            D  |  4  |  5  |    |
                                            ---------------------
                                            .5 |     |     |    | vmovups xmm1, xmmword ptr [r12+r14*4+0x10]
                                            .5 |     |     |    | vmovups xmm0, xmmword ptr [rsi+r14*4+0x10]
                                            .5 |     |     |    | vmovups xmm13, xmmword ptr [r12+r14*4+0x14]
                                            .5 |     |     |    | vmovups xmm14, xmmword ptr [r12+r14*4+0xc]
                                            .5 |     |     |    | vmovups xmm12, xmmword ptr [rdi+r14*4+0x10]
                                            .5 |     |     |    | mov r15, qword ptr [rsp+0x338]
                                            .5 |     | 1.0 |    | vinsertf128 ymm10, ymm1, xmmword ptr [r12+r
                                               |     |     |    | vaddps ymm1, ymm7, ymm10
                                               |     |     |    | vaddps ymm15, ymm8, ymm10
                                            .5 |     | 1.0 |    | vinsertf128 ymm11, ymm0, xmmword ptr [rsi+r
                                               |     |     |    | vaddps ymm1, ymm1, ymm11
```

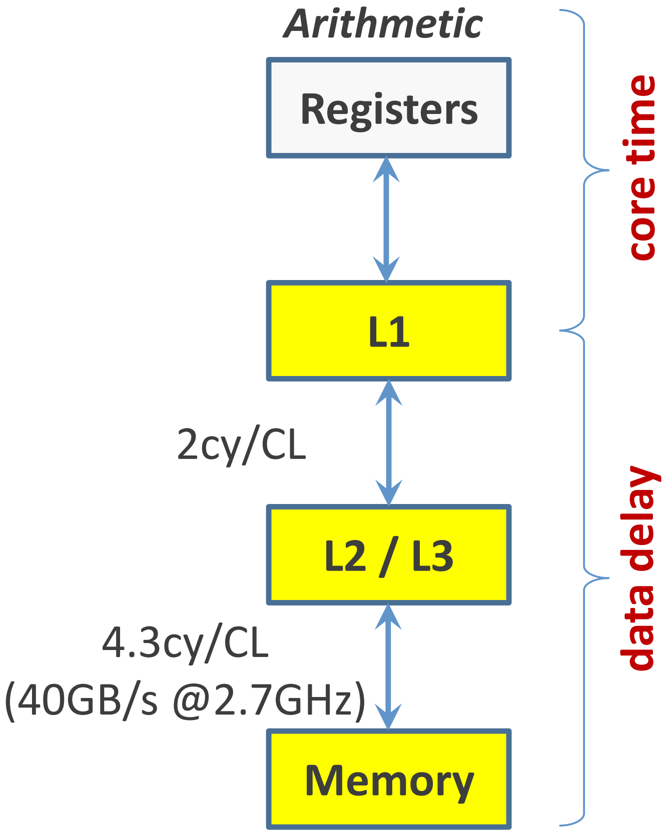


Execution-Cache-Memory (ECM) model – Basics

Single core execution time is determined by

- **Core** execution **time &**
- **Data delay** in memory hierarchy

Socket scaling is linear
until relevant shared bottleneck is hit



Insights

- Single-core performance & socket scaling
- Relevant bottlenecks & performance impact

Input

- Full socket STREAM bandwidth
- All data transfers in all memory levels (**Machine ←→ Application**)
- Unit of work: 1 cache line's "worth of work"

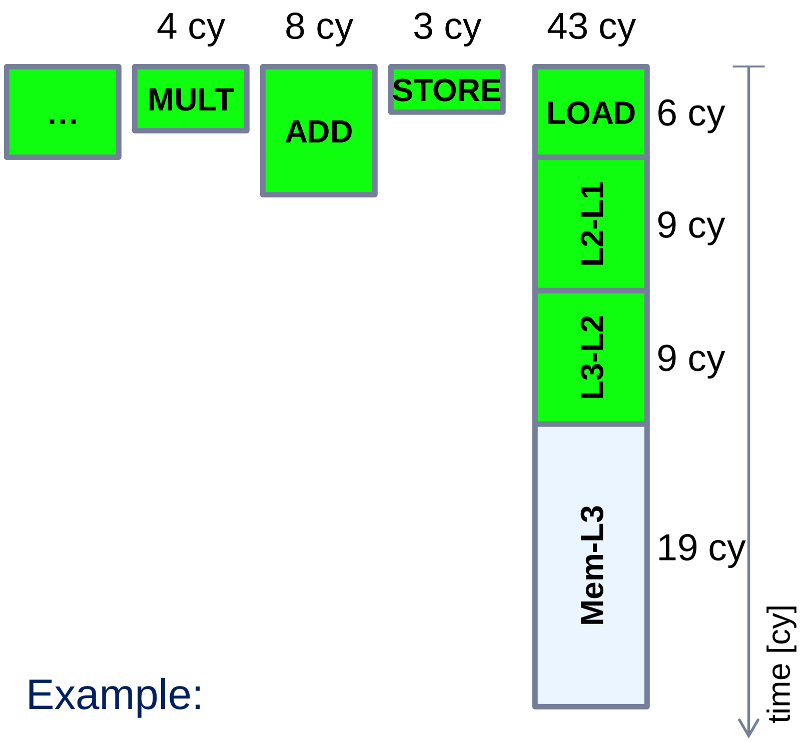
ECM model – the rules

1. LOADs in the L1 cache do not overlap with any other data transfer in the memory hierarchy

2. Everything else in the core overlaps perfectly with data transfers (STOREs show some non-overlap)

3. The scaling limit is set by the ratio of

$$\frac{\#\text{ cycles per CL overall}}{\#\text{ cycles per CL at the bottleneck}}$$



Example:

Single-core (data in L1): 8 cy (ADD)

Single-core (data in memory):
6+9+9+19 cy = 43 cy

Scaling limit: 43 / 19 = 2.3 cores

ECM model – composition

ECM predicted time

T_{ECM} = maximum of overlapping time and sum of all other contributions

$$T_{core} = \max(T_{nOL}, T_{OL})$$

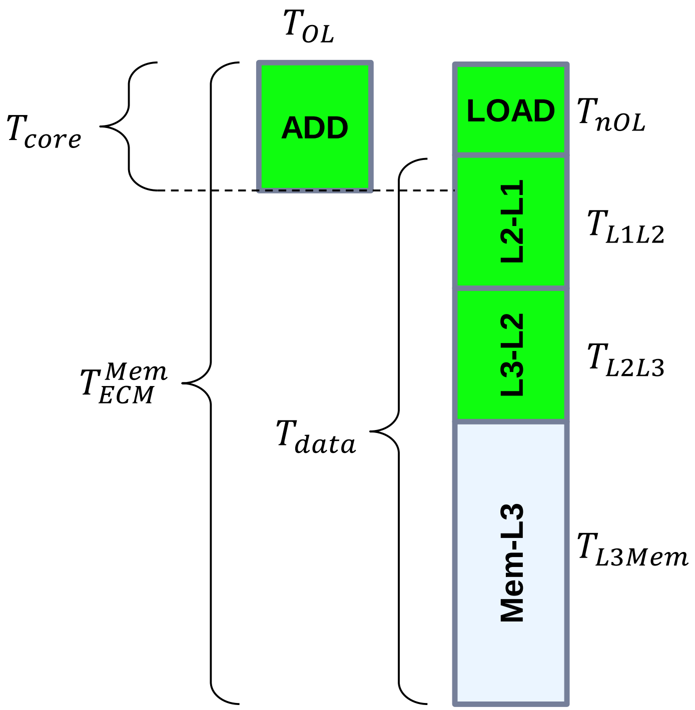
$$\boxed{T_{ECM} = \max(T_{nOL} + T_{data}, T_{OL})}$$

Shorthand notation for time contributions:

$$\{ \underbrace{T_{OL} \parallel T_{nOL} \mid T_{L1L2} \mid T_{L2L3}}_{\text{\# cy invariant to clock speed}} \mid \underbrace{T_{L3Mem}}_{\text{\# cy changes w/ clock speed}} \}$$

Example from previous slide:

$$\{ 8 \parallel 6 \mid 9 \mid 9 \mid 19 \}\ \mathrm{cy}$$



ECM model – prediction

Notation for cycle predictions in different memory hierarchy levels:

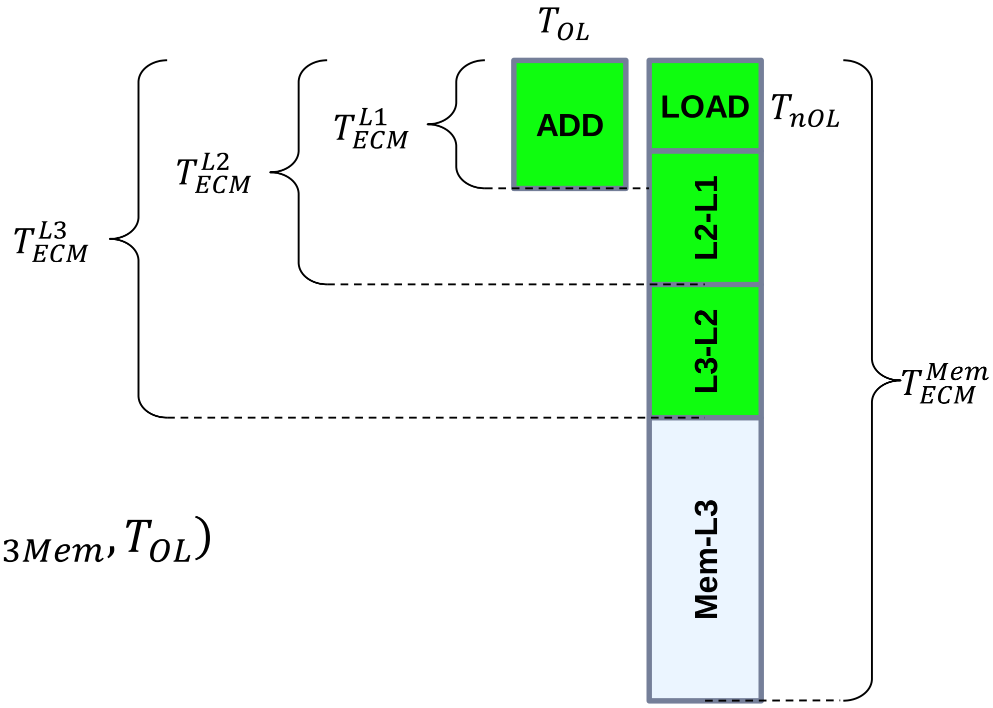
$$\{\, T_{ECM}^{L1} \,\rceil\, T_{ECM}^{L2} \,\rceil\, T_{ECM}^{L3} \,\rceil\, T_{ECM}^{Mem} \,\}$$

$$T_{ECM}^{L1} = T_{core} = \max(T_{nOL}, T_{OL})$$

$$T_{ECM}^{L2} = \max(T_{nOL} + T_{L1L2}, T_{OL})$$

$$T_{ECM}^{L3} = \max(T_{nOL} + T_{L1L2} + T_{L2L3}, T_{OL})$$

$$T_{ECM}^{Mem} = \max(T_{nOL} + T_{L1L2} + T_{L2L3} + T_{L3Mem}, T_{OL})$$



Example: $\{\, 8 \,\rceil\, 15 \,\rceil\, 24 \,\rceil\, 43 \,\}$ cy

Experimental data (measured) notation: $8.6 \,\rceil\, 16.2 \,\rceil\, 26 \,\rceil\, 47$ cy

ECM model – from time to performance

f_0: base clock speed [cy/s]
f: actual cock speed [cy/s]

Performance is work (W) over time:

$$P_{ECM} = \frac{W \cdot f}{\{ \underbrace{T^{L1}_{ECM} \rceil T^{L2}_{ECM} \rceil T^{L3}_{ECM}}_{\text{in-cache performance is proportional to } f} \rceil \max(T^{L3}_{ECM} + T_{L3Mem} \cdot f/f_0 \,, T_{OL})\}}$$

ratio T^{L3}_{ECM}/T_{L3Mem} quantifies f-sensitivity of serial in-memory performance

Example:

$$P = \frac{32 \text{ flops} \cdot f}{\{ 8 \rceil 15 \rceil 24 \rceil 24 + 19 \cdot f/f_0 \} \text{ cy}}$$

ECM model – saturation

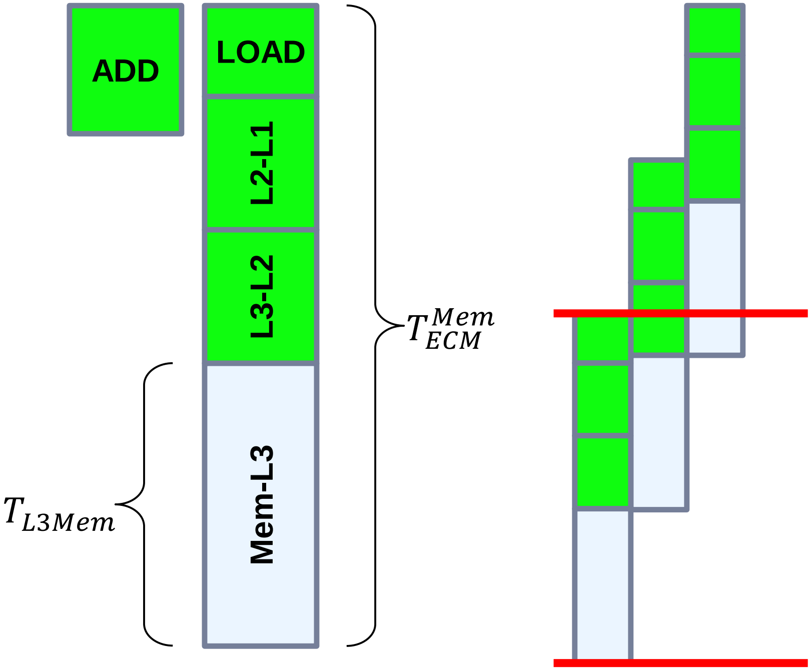
Main assumption: Performance scaling is linear until a bandwidth bottleneck (b_S) is hit

Performance vs. cores (Memory BN):

$$P_{ECM}(n) = \min\left(nP_{ECM}^{Mem}, \frac{b_S^{Mem}}{B_C^{Mem}}\right)$$

Number of cores at saturation:

$$n_S = \left\lceil \frac{b_S/B_C}{P_{ECM}^{Mem}} \right\rceil = \left\lceil \frac{T_{ECM}^{Mem}}{T_{L3Mem}} \right\rceil$$



Example:

$$\{\,8 \,\|\, 6 \,|\, 9 \,|\, 9 \,|\, 19\,\}\ \text{cy}, \qquad \{\,8 \,]\, 15 \,]\, 24 \,]\, 43\,\}\ \text{cy} \implies n_S = \left\lceil \frac{43}{19} \right\rceil = 3$$

ECM vs. Roofline

Roofline assumes **full overlap** of all execution and transfer times

Roofline requires **measured baseline bandwidth limits** for all memory levels i (L2…Memory) at all core counts n: $b_S^i(n)$

$$\left.\begin{aligned} T_{Roof}^{L1} &= T_{core} = \max(T_{nOL}, T_{OL}) \\ &\vdots \\ T_{Roof}^{Mem} &= \max(T_{nOL}, T_{L1L2}, T_{L2L3}, T_{L3Mem}, T_{OL}) \end{aligned}\right\} P_{Roof}(n) = \min_i\left(nP_{ECM}^{L1}, \frac{b_S^i(n)}{B_C^i}\right)$$

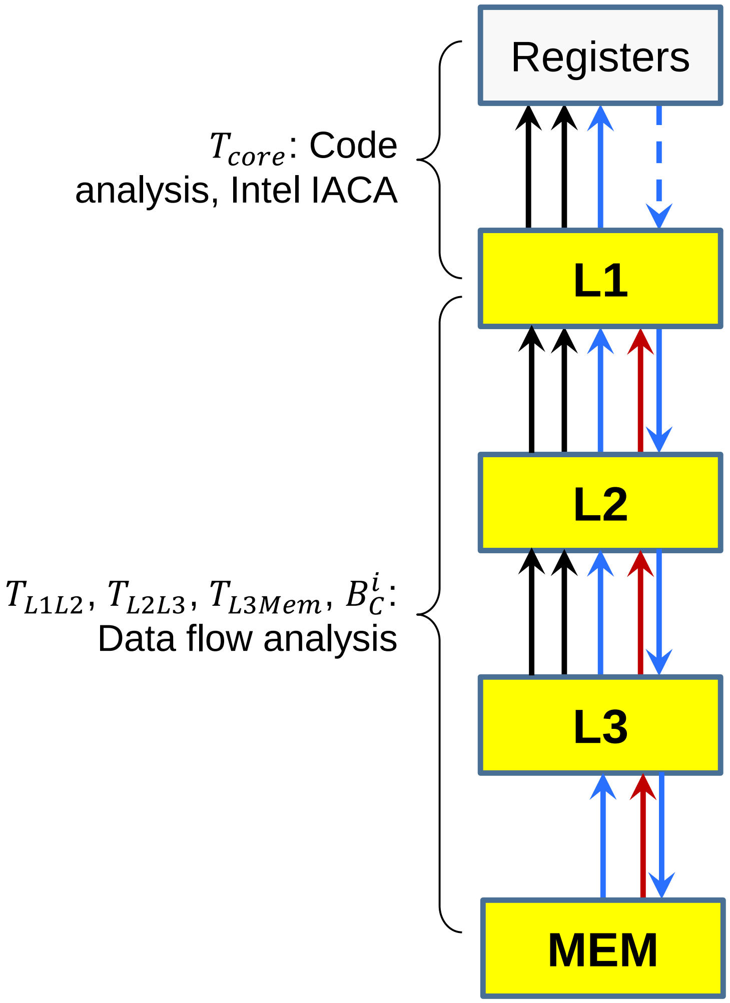
Roofline ≈ ECM if:

- $T_{core} \gg T_{data}$: non-overlapping data transfers are insignificant
  or
- Loop kernel is similar to streaming benchmark used to obtain $b_S^i(n)$

How do we get the numbers?

Basic information about hardware capabilities:

- In-core limitations
  - Throughput limits:µops, LD/ST, ADD/MULT per cycle
  - Pipeline depths
- Cache hierarchy
  - **ECM**: Cycles per CL transfer
  - **RL**: measured max bandwidths for all cache levels, core counts
- Memory interface
  - **ECM**: measured saturated BW
  - **RL**: measured max bandwidths for all core counts



T_{core}: Code analysis, Intel IACA

$T_{L1L2}, T_{L2L3}, T_{L3Mem}, B_C^i$: Data flow analysis

WARMUP: ARRAY SUMMATION

```
for(i=0; i<N; ++i)
   s += a[i];
```

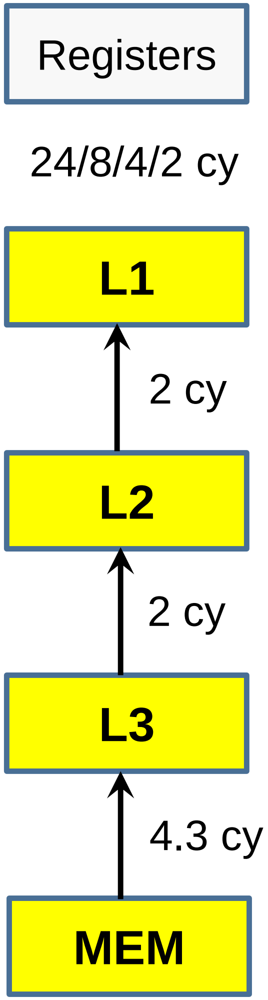
Unit of work (1 CL): 8 flops

Data transfer per unit: 1 CL

ECM Model for array sum on SNB 2.7 GHz

```
for(i=0; i<N; ++i)
  s += a[i];
```

<table>
<tr><th>case</th><th>ECM Model [cy]</th><th>prediction [cy]</th><th>measurement [cy]</th></tr>
<tr><td>naive</td><td>{24 ‖ 4 | 2 | 2 | 4.3}</td><td>{24 ⌉ 24 ⌉ 24 ⌉ 24}</td><td>24 ⌉ 24 ⌉ 24 ⌉ 27</td></tr>
<tr><td>scalar</td><td>{8 ‖ 4 | 2 | 2 | 4.3}</td><td>{8 ⌉ 8 ⌉ 8 ⌉ 12}</td><td>8.1 ⌉ 8.9 ⌉ 11 ⌉ 18</td></tr>
<tr><td>SSE</td><td>{4 ‖ 2 | 2 | 2 | 4.3}</td><td>{4 ⌉ 4 ⌉ 6 ⌉ 10}</td><td>4.1 ⌉ 5 ⌉ 7.2 ⌉ 15</td></tr>
<tr><td>AVX</td><td>{2 ‖ 2 | 2 | 2 | 4.3}</td><td>{2 ⌉ 4 ⌉ 6 ⌉ 10}</td><td>2.1 ⌉ 4.9 ⌉ 7.2 ⌉ 14</td></tr>
</table>



Naive = scalar, no unrolling (full 3 cy penalty per ADD)

Scalar = scalar, 4-way modulo unrolling

SSE = 4-way modulo unrolling + 2-way SIMD

AVX = 4-way modulo unrolling + 4-way SIMD

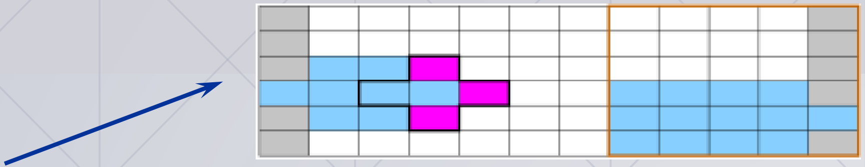
2D 5-PT JACOBI STENCIL (DOUBLE PRECISION)

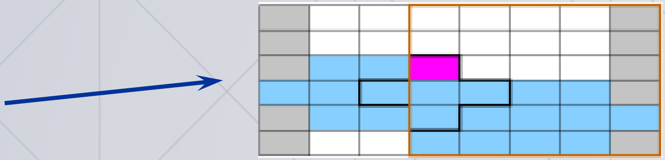
```
for(j=1; j < Nj-1; ++j)
 for(i=1; i < Ni-1; ++i)
  b[j][i] = (a[ j ][i-1] + a[ j ][i+1]
           + a[j-1][ i ] + a[j+1][ i ] ) * s;
```

Unit of work (1 CL): 8 LUPs

Data transfer per unit:

- 5 CL if layer condition violated in higher cache level
- 3 CL if layer condition satisfied





ECM Model for 2D Jacobi (AVX) on SNB 2.7 GHz

Radius-r stencil → $(2r+1)$ layers have to fit

```
for(j=1; j < Nj-1; ++j)
 for(i=1; i < Ni-1; ++i)
  b[j][i] = (a[ j ][i-1] + a[ j ][i+1]
                + a[j-1][ i ] + a[j+1][ i ] ) * s;
```

Cache k has size C_k

Layer condition:

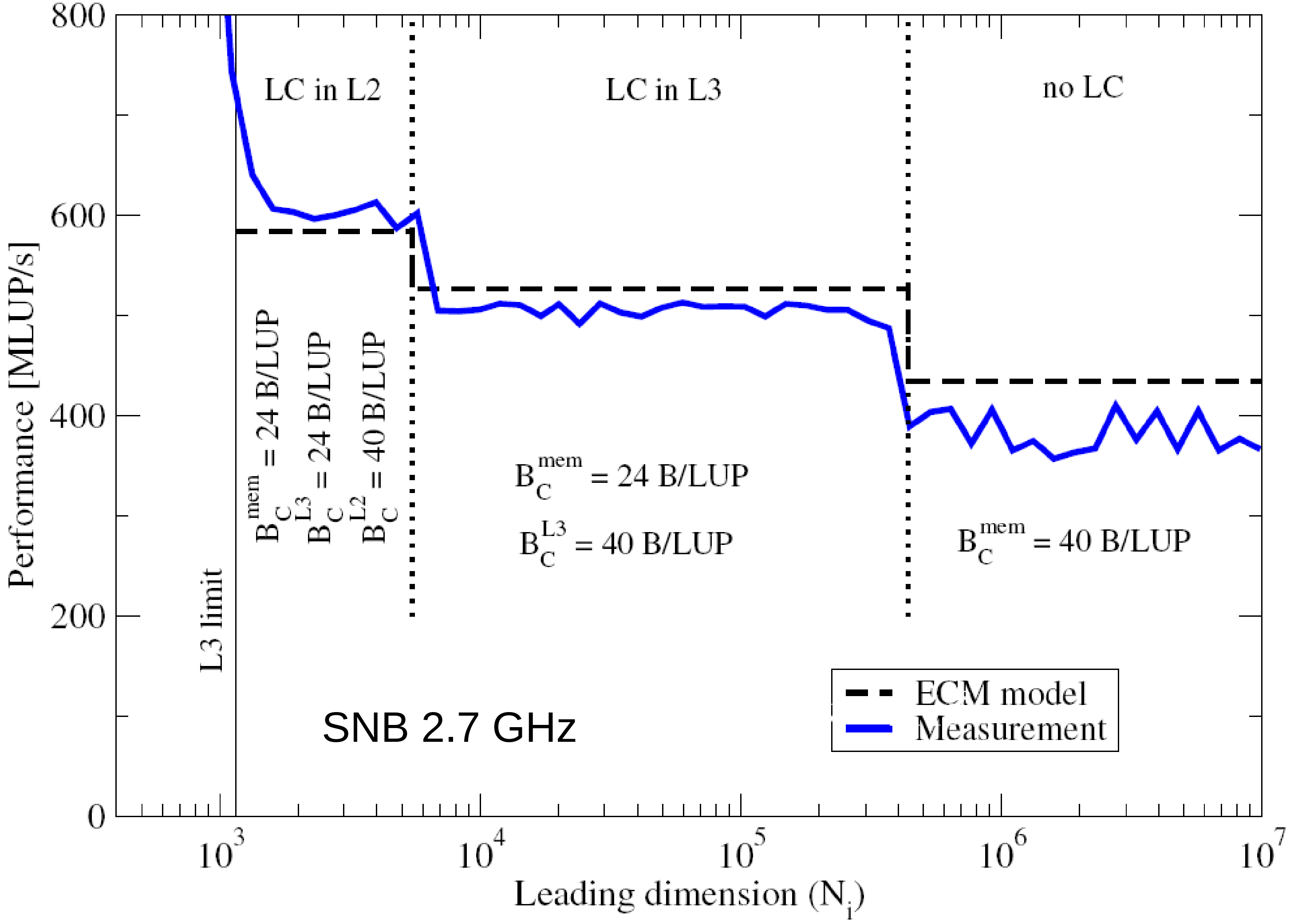
$$(2r+1) \cdot N_i \cdot 8\,\mathrm{B} < \frac{C_k}{2}$$

2D 5-pt: $r = 1$

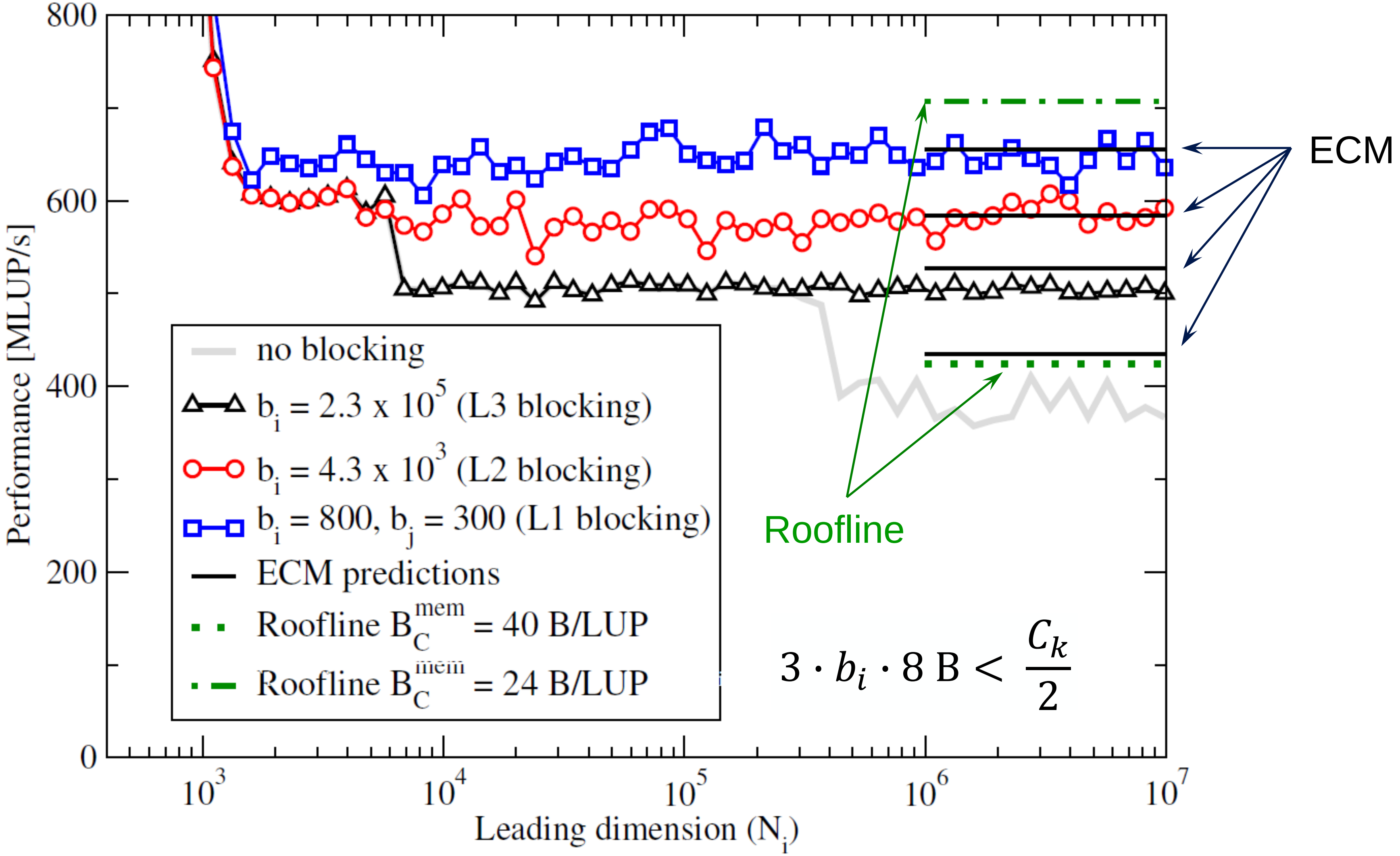
LC	ECM Model [cy]	prediction [cy]	$P_{\mathrm{ECM}}^{\mathrm{mem}}$ [MLUPS]	$N_i <$	n_S
L1	$\{6 \Vert 8 \vert 6 \vert 6 \vert 13\}$	$\{8 \rceil 14 \rceil 20 \rceil 33\}$	659	683	3
L2	$\{6 \Vert 8 \vert 10 \vert 6 \vert 13\}$	$\{8 \rceil 18 \rceil 24 \rceil 37\}$	587	5461	3
L3	$\{6 \Vert 8 \vert 10 \vert 10 \vert 13\}$	$\{8 \rceil 18 \rceil 28 \rceil 41\}$	529	436900	4
—	$\{6 \Vert 8 \vert 10 \vert 10 \vert 22\}$	$\{8 \rceil 18 \rceil 28 \rceil 50\}$	438	N/A	3

LC = layer condition satisfied in …

2D 5-pt serial in-memory performance and layer conditions



2D 5-pt: impact of inner loop blocking on SNB

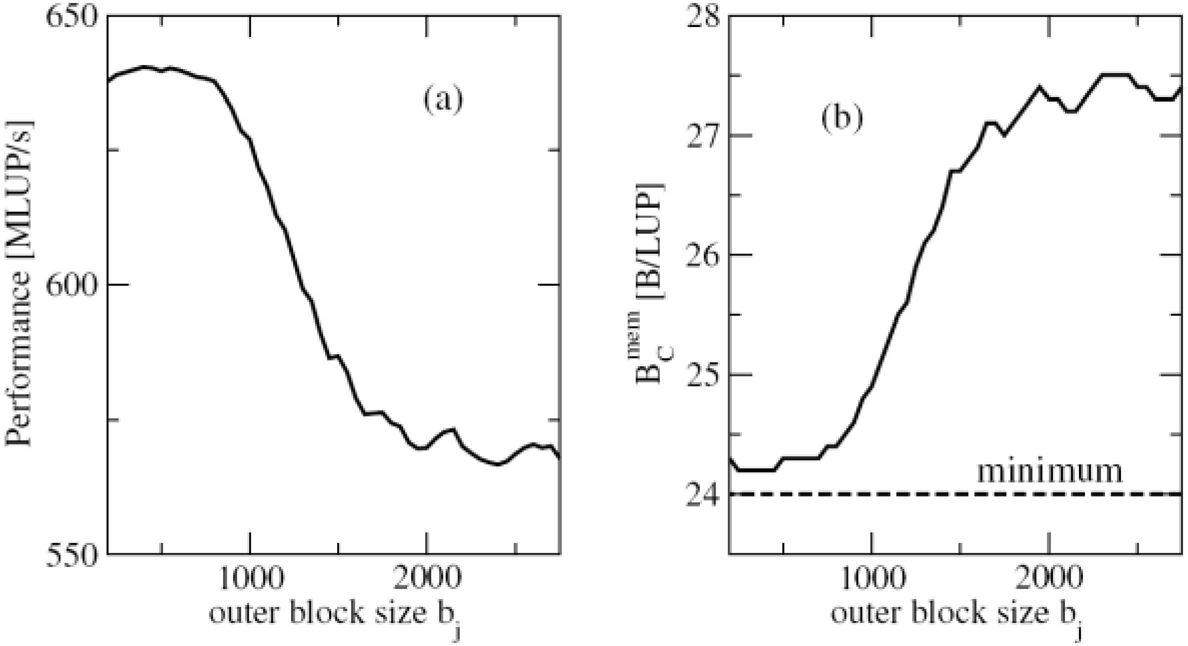


ECM

Roofline

$$3 \cdot b_i \cdot 8\,\mathrm{B} < \frac{C_k}{2}$$

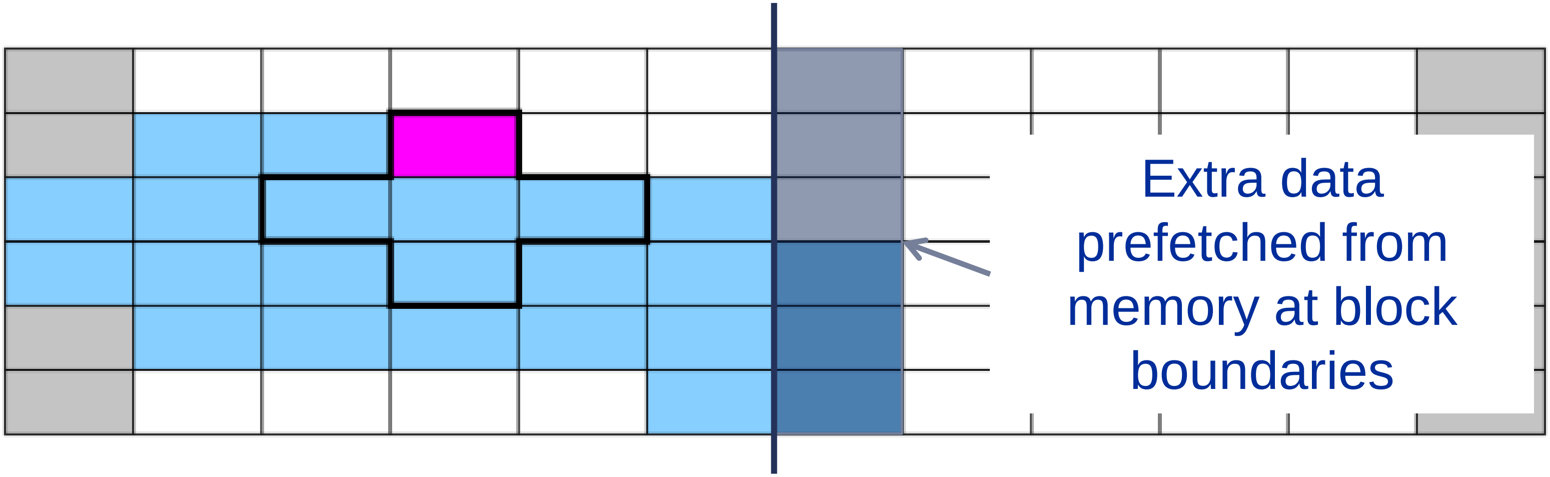
2D 5-pt: Why outer loop blocking?



$$b_i = 800$$

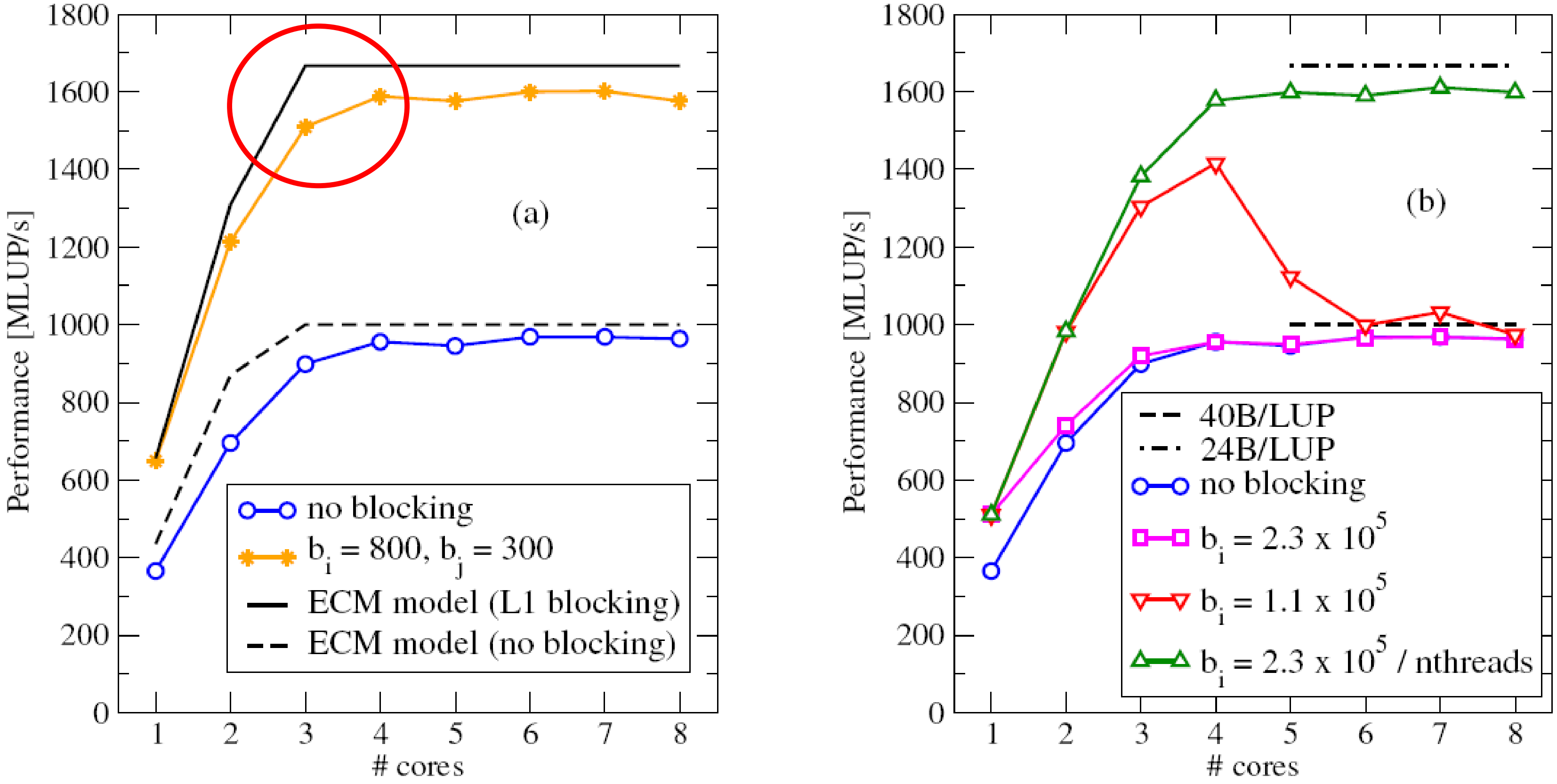
$$N_i = 3.5 \times 10^4$$

$$N_j = 1.2 \times 10^4$$



Extra data prefetched from memory at block boundaries

2D 5-pt multi-core scaling

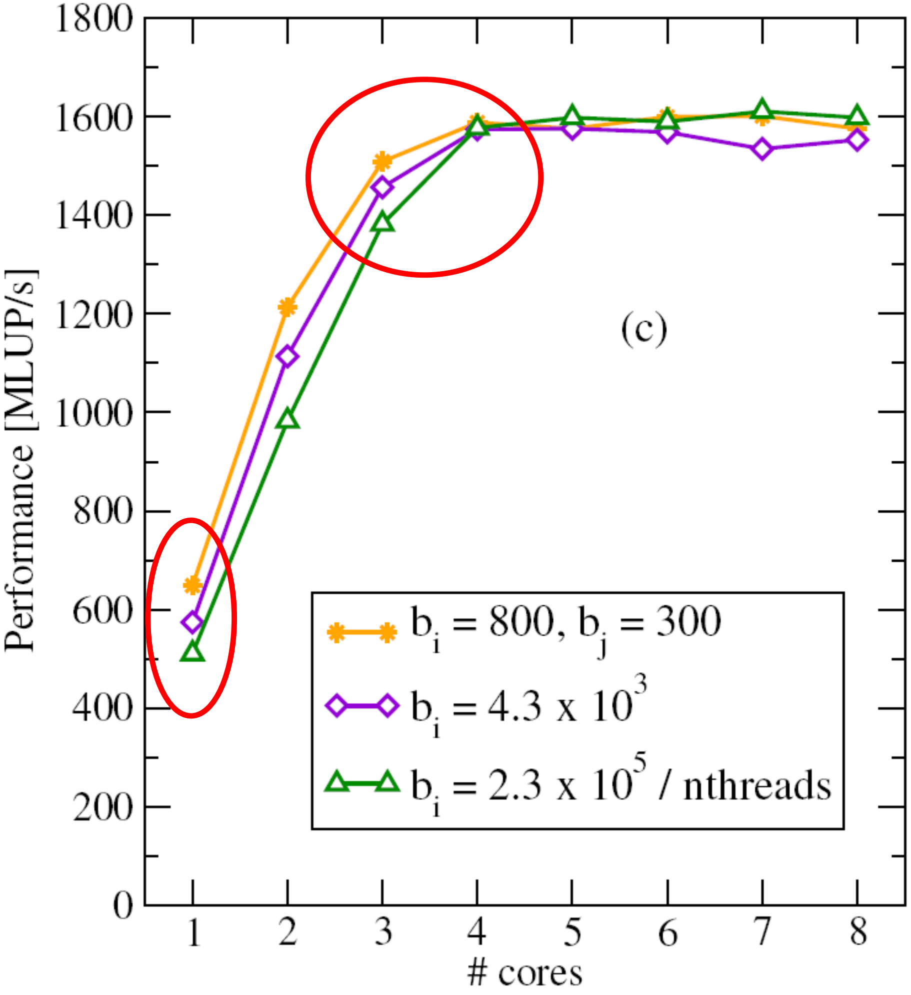


Modified layer condition for static work-sharing of outer loop:

$$(2r + 1) \cdot b_i \cdot n_{threads} \cdot 8\ \mathrm{B} < \frac{C_k}{2}$$

2D 5-pt spatial blocking variants & saturation

LC	$P_{\mathrm{ECM}}^{\mathrm{mem}}$ [MLUPS]
L1	659
L2	587
L3	529
—	438



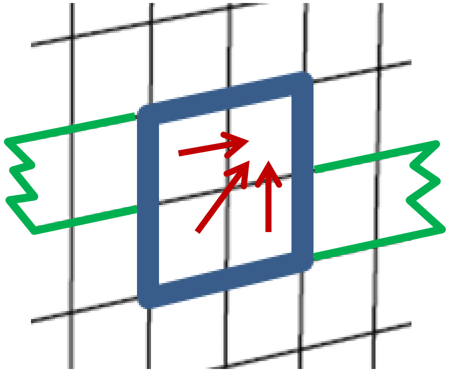
3D RANGE-2 STENCIL „UXX“ (SINGLE & DOUBLE PRECISION)

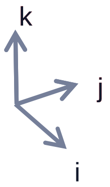
```
#pragma omp parallel for private(d) schedule(static)
for(int k=2; k<=N-1; k++){
  for(int j=2; j<=N-1; j++){
    for (int i=2; i<=N-1; i++){
      d = 0.25*(d1[ k ][j][i] + d1[ k ][j-1][i]
              + d1[k-1][j][i] + d1[k-1][j-1][i]);
      u1[k][j][i] = u1[k][j][i] + (dth/d)
             *( c1 *(xx[ k ][ j ][ i ]-xx[ k ][ j ][i-1])
              + c2 *(xx[ k ][ j ][i+1]-xx[ k ][ j ][i-2])
              + c1 *(xy[ k ][ j ][ i ]-xy[ k ][j-1][ i ])
              + c2 *(xy[ k ][j+1][ i ]-xy[ k ][j-2][ i ])
              + c1 *(xz[ k ][ j ][ i ]-xz[k-1][ j ][ i ])
              + c2 *(xz[k+1][ j ][ i ]-xz[k-2][ j ][ i ]));
}}}}
```

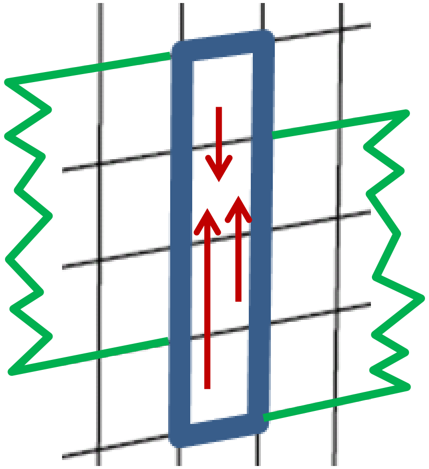
Uxx stencil layer conditions (outer levels only)

`d1[0;-1][0;-1][*]`

`xz[-2;+1][*][*]`







→ 2 **`d1`** layers

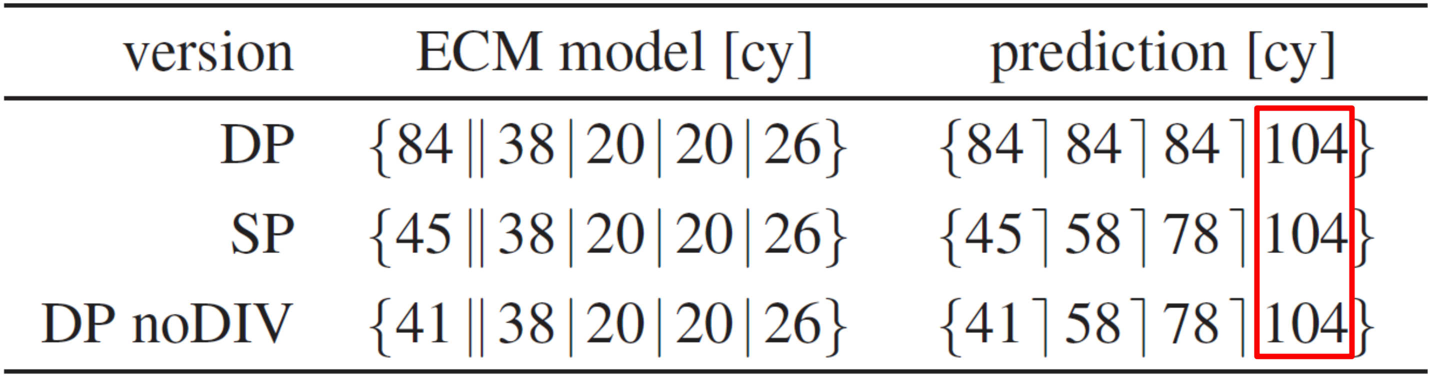
→ 4 **`xz`** layers

4+2 layers must fit

Uxx stencil ECM analysis

Apply L3 blocking of j loop according to layer condition (problem size $N_i \times N_j \times N_k$):

$$(4+2) \cdot N_i \cdot b_j \cdot n_{threads} \cdot \left\{ \begin{array}{c} 4\text{ B (SP)} \\ 8\text{ B (DP)} \end{array} \right\} < \frac{C_3}{2}$$

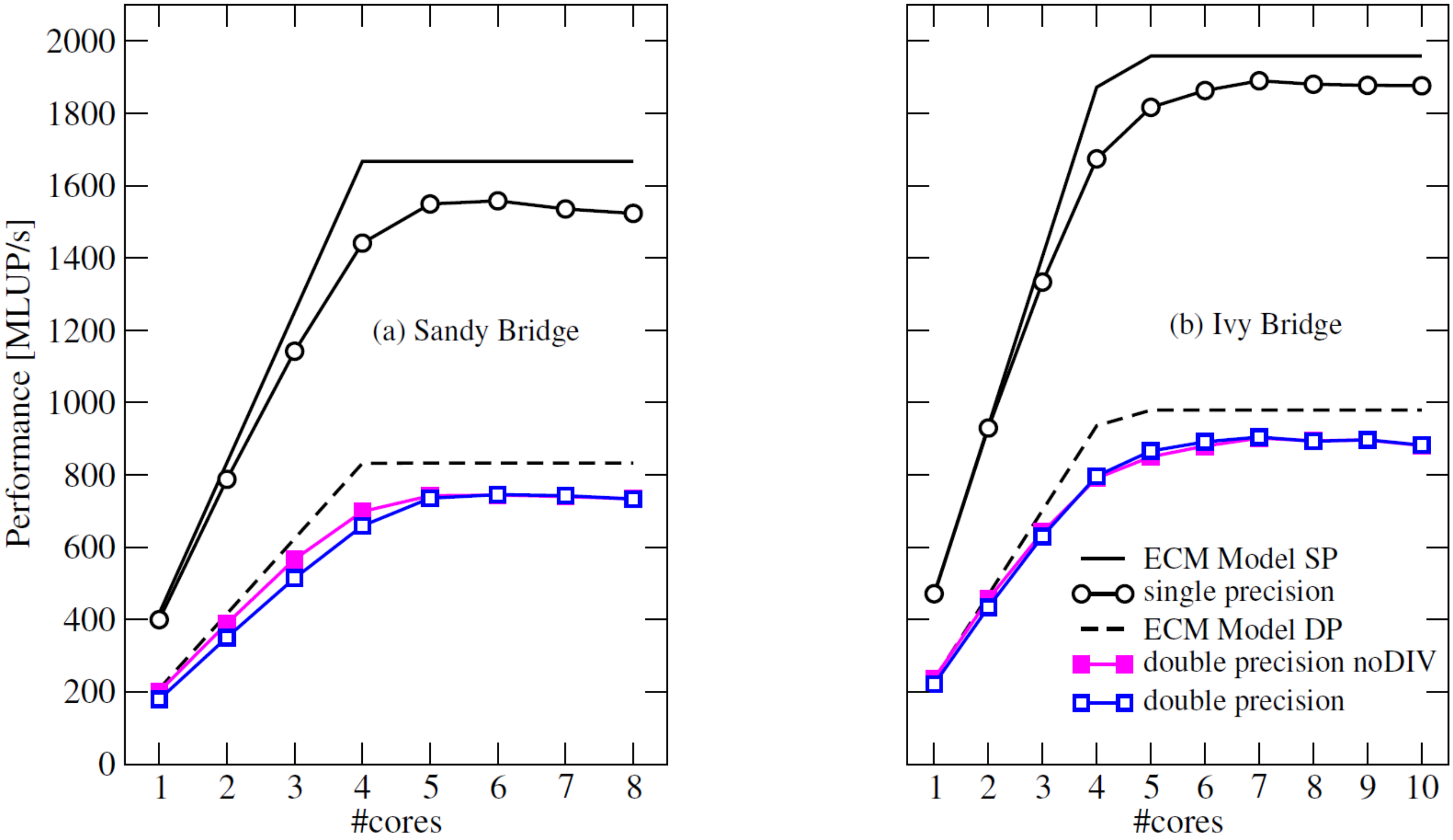


version	ECM model [cy]	prediction [cy]
DP	$\{84 \,\Vert\, 38 \,\vert\, 20 \,\vert\, 20 \,\vert\, 26\}$	$\{84 \rceil 84 \rceil 84 \rceil 104\}$
SP	$\{45 \,\Vert\, 38 \,\vert\, 20 \,\vert\, 20 \,\vert\, 26\}$	$\{45 \rceil 58 \rceil 78 \rceil 104\}$
DP noDIV	$\{41 \,\Vert\, 38 \,\vert\, 20 \,\vert\, 20 \,\vert\, 26\}$	$\{41 \rceil 58 \rceil 78 \rceil 104\}$

Consequences:

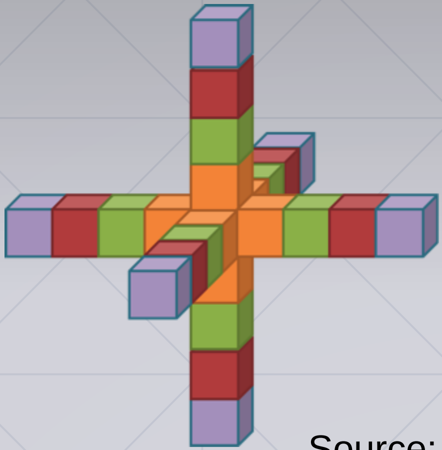
- No use in removing DIV from loop in DP for in-memory case
- No actual DIV in code for SP (compiler employs rcpps + NR)
- Temporal blocking not much use for serial, but makes the code scalable!

Uxx performance and scaling on SNB and IVB



3D LONG-RANGE STENCIL (SINGLE PRECISION)

```
#pragma omp parallel for
for(int k=4; k < N-4; k++) {
 for(int j=4; j < N-4; j++) {
  for(int i=4; i < N-4; i++) {
   float lap = c0 * %V%[k][j][i]
  + c1 * ( V[ k ][ j ][i+1]+ V[ k ][ j ][i-1])
  + c1 * ( V[ k ][j+1][ i ]+ V[ k ][j-1][ i ])
  + c1 * ( V[k+1][ j ][ i ]+ V[k-1][ j ][ i ])
    ...
  + c4 * ( V[ k ][ j ][i+4]+ V[ k ][ j ][i-4])
  + c4 * ( V[ k ][j+4][ i ]+ V[ k ][j-4][ i ])
  + c4 * ( V[k+4][ j ][ i ]+ V[k-4][ j ][ i ]);
   U[k][j][i] = 2.f * V[k][j][i] - U[k][j][i]
              + ROC[k][j][i] * lap;
}}}
```



Source: http://goo.gl/dqOlnI

3D long-range SP stencil ECM model

Layer condition in L3 at problem size $N_i \times N_j \times N_k$:

$$9 \cdot N_i \cdot b_j \cdot n_{threads} \cdot 4\,\mathrm{B} < \frac{C_3}{2}$$

ECM Model: $\{\, 68 \,\|\, 62 \mid 24 \mid 24 \mid 17 \,\}\ \mathrm{cy}$ → $\{\, 68 \rceil 86 \rceil 110 \rceil 127 \,\}\ \mathrm{cy}$

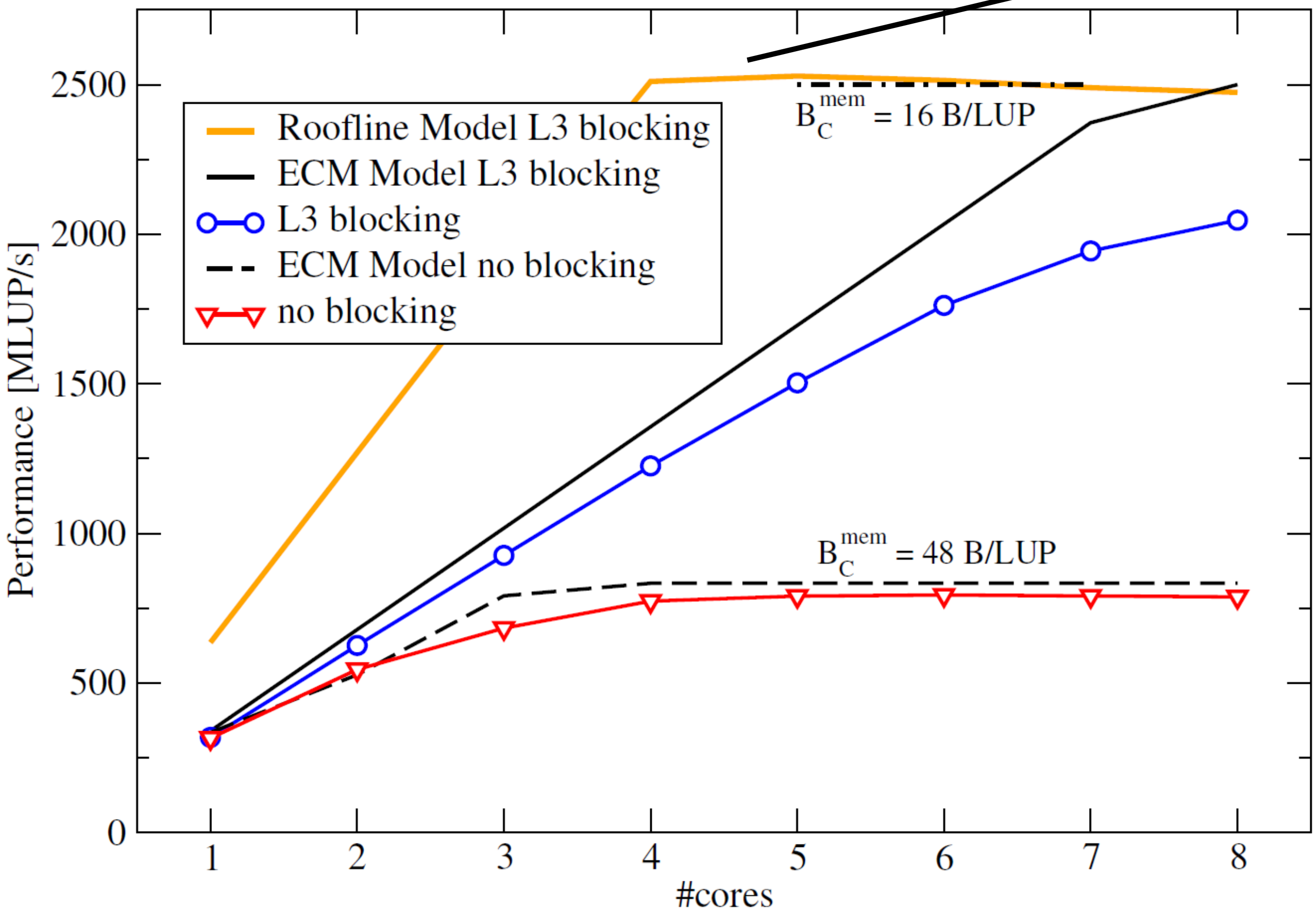
T_{L3Mem} plays minor part

Saturation at $n_s = \left\lceil \frac{127}{17} \right\rceil = 8$ cores.

Consequences:

- Temporal blocking will not yield substantial speedup
- Improve low-level code first (semi-stencil…?)

3D long-range SP stencil results (SNB)

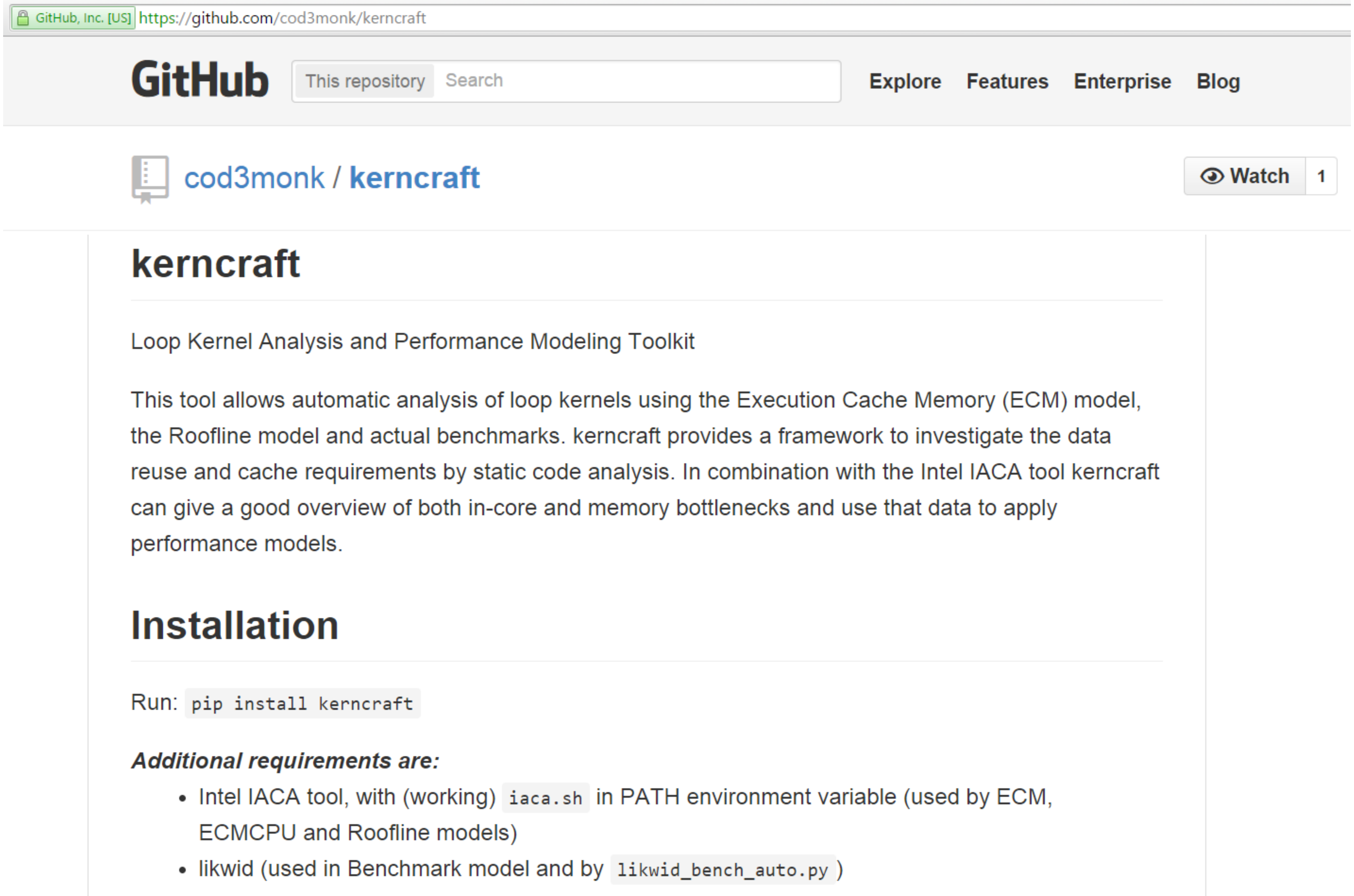


Roofline too optimistic due to overlapping assumption

KERNCRAFT

First steps towards automated model construction

kerncraft: ECM/Roofline modeling toolkit



kerncraft

Loop Kernel Analysis and Performance Modeling Toolkit

This tool allows automatic analysis of loop kernels using the Execution Cache Memory (ECM) model, the Roofline model and actual benchmarks. kerncraft provides a framework to investigate the data reuse and cache requirements by static code analysis. In combination with the Intel IACA tool kerncraft can give a good overview of both in-core and memory bottlenecks and use that data to apply performance models.

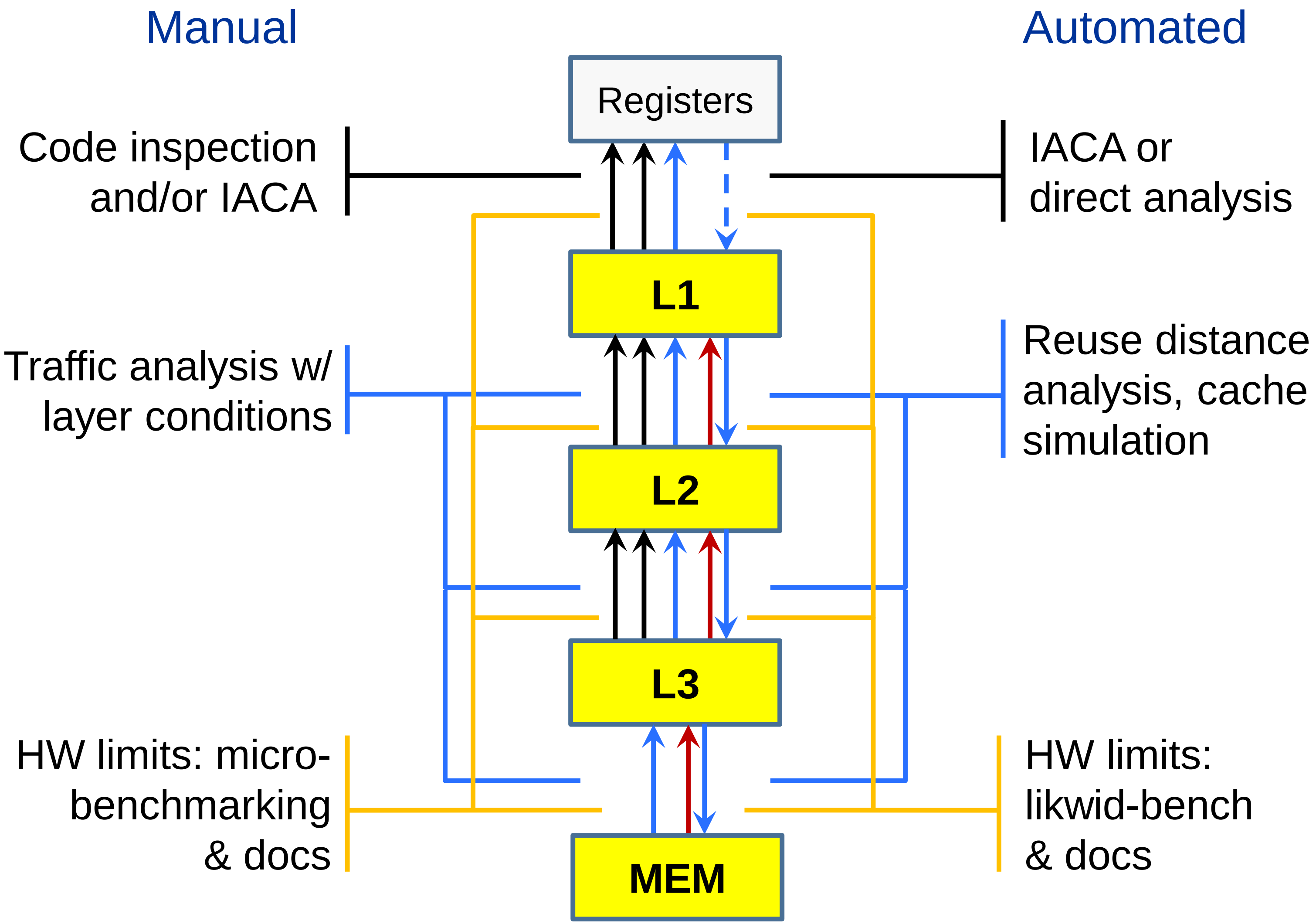
Installation

Run: `pip install kerncraft`

Additional requirements are:

- Intel IACA tool, with (working) `iaca.sh` in PATH environment variable (used by ECM, ECMCPU and Roofline models)
- likwid (used in Benchmark model and by `likwid_bench_auto.py`)

Towards automated model generation



Manual

Automated

Registers

L1

L2

L3

MEM

Code inspection
and/or IACA

Traffic analysis w/
layer conditions

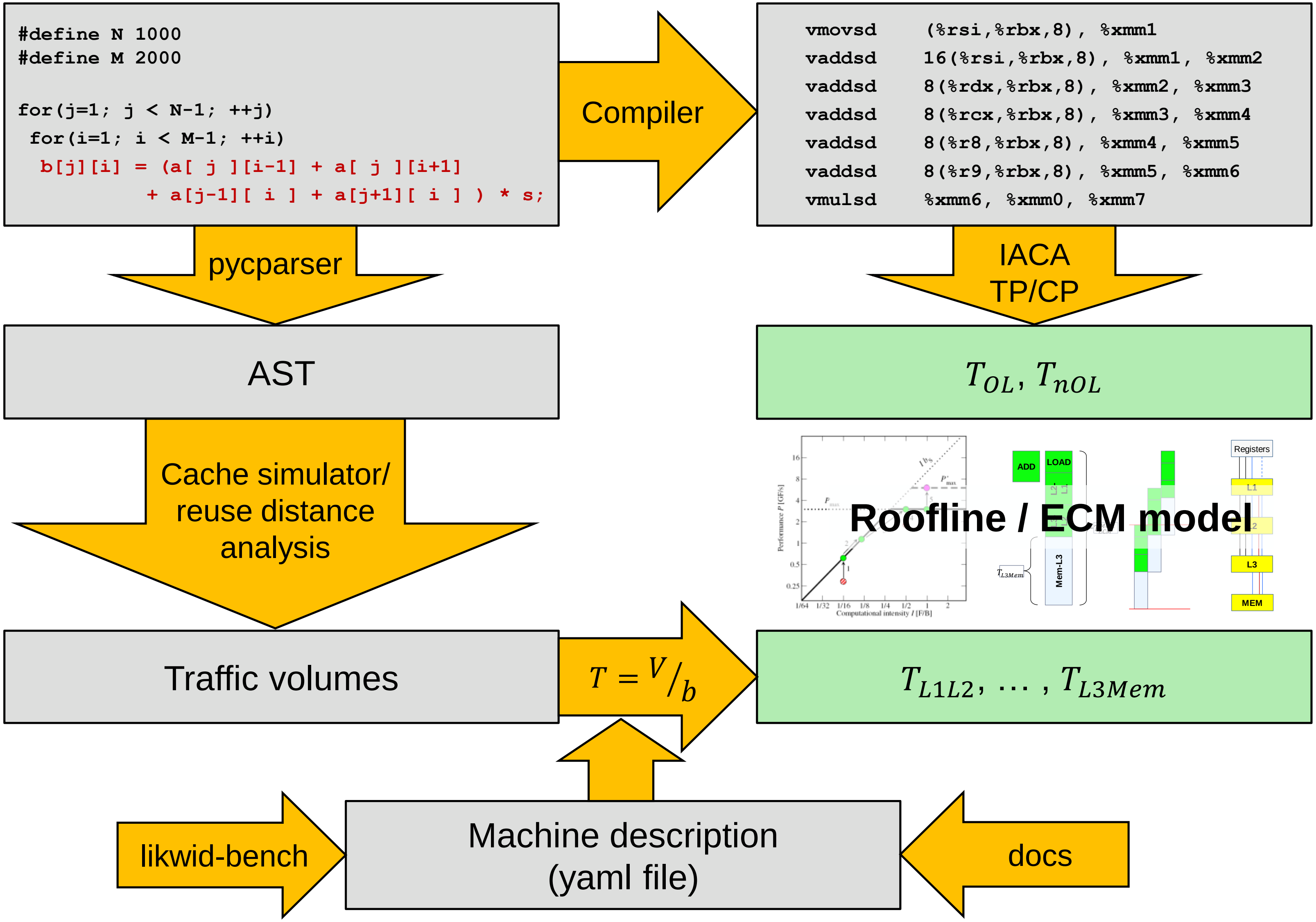
HW limits: micro-
benchmarking
& docs

IACA or
direct analysis

Reuse distance
analysis, cache
simulation

HW limits:
likwid-bench
& docs

kerncraft



```
#define N 1000
#define M 2000

for(j=1; j < N-1; ++j)
 for(i=1; i < M-1; ++i)
  b[j][i] = (a[ j ][i-1] + a[ j ][i+1]
           + a[j-1][ i ] + a[j+1][ i ] ) * s;
```

Compiler

```
vmovsd    (%rsi,%rbx,8), %xmm1
vaddsd    16(%rsi,%rbx,8), %xmm1, %xmm2
vaddsd    8(%rdx,%rbx,8), %xmm2, %xmm3
vaddsd    8(%rcx,%rbx,8), %xmm3, %xmm4
vaddsd    8(%r8,%rbx,8), %xmm4, %xmm5
vaddsd    8(%r9,%rbx,8), %xmm5, %xmm6
vmulsd    %xmm6, %xmm0, %xmm7
```

pycparser

IACA TP/CP

AST

T_{OL}, T_{nOL}

Cache simulator/ reuse distance analysis

Roofline / ECM model

Traffic volumes

$T = {}^{V}/_{b}$

$T_{L1L2}, \ldots, T_{L3Mem}$

likwid-bench

Machine description (yaml file)

docs

kerncraft example (ECM)

```
$ kerncraft -vv -p ECM -m phinally.yaml 2d-5pt.c -D N 10000 -D M 10000
================================================================================
                                    2d-5pt.c
================================================================================
double a[M][N];
double b[M][N];
double s;

for(int j=1; j<M-1; ++j)
    for(int i=1; i<N-1; ++i)
        b[j][i] = ( a[j][i-1] + a[j][i+1]
                  + a[j-1][i] + a[j+1][i]) * s;

variables:     name |   type size
           ---------+--------------------------
                  a | double (10000, 10000)
                  s | double None
                  b | double (10000, 10000)
```

kerncraft example (ECM) continued

```
loop stack:             idx |        min        max       step
                   ---------+---------------------------------
                          j |          1       9999         +1
                          i |          1       9999         +1

data sources:          name |  offsets   ...
                   ---------+------------...
                          a | ('rel', 'j', 0), ('rel', 'i', -1)
                            | ('rel', 'j', 0), ('rel', 'i', 1)
                            | ('rel', 'j', -1), ('rel', 'i', 0)
                            | ('rel', 'j', 1), ('rel', 'i', 0)
                          s | ('dir',)

data destinations:     name |  offsets   ...
                   ---------+------------...
                          b | ('rel', 'j', 0), ('rel', 'i', 0)
```

kerncraft example (ECM) continued

```
FLOPs:      op | count
           ----+-------
             + |    3
             * |    1
                =======
                    4

constants:      name | value
           ---------+----------
                   M | 10000
                   N | 10000


Ports and cycles: {'1': 6.0, '0DV': 0.0, '2D': 8.0, '0': 5.05, '3': 9.0, '2':
9.0, '5': 5.95, '4': 4.0, '3D': 8.0}
Uops: 37.0
Throughput: 9.45cy per CL
T_nOL = 8.0cy
T_OL = 9.0cy
```

kerncraft example (ECM) continued

```
Trace length per access in L1: 982
Hits in L1: 30 {'a': {'ji': [10006, 10005, 10004, 10003, 10002, 10001, 10000,
7, 6, 5, 4, 3, 2, 1, 0, -1, -9994, -9995, -9996, -9997, -9998, -9999, -10000]},
's': {}, 'b': {'ji': [6, 5, 4, 3, 2, 1, 0]}}
Misses in L1: 4 (4CL): {'a': {'ji': [10007, 8, -9993]}, 's': {}, 'b': {'ji':
[7]}}
Evicts from L1 8 (1CL): {'a': {}, 's': {}, 'b': {'ji': [7, 6, 5, 4, 3, 2, 1,
0]}}

...

L1-L2 = 10cy
L2-L3 = 10cy
L3-MEM = 12.96cy
{ 9.0 || 8.0 | 10 | 10 | 12.96 } = 40.96cy
```

kerncraft example (Roofline)

```
$ kerncraft -v -p Roofline -m phinally.yaml 2d-5pt.c -D N 10000 -D M 10000

...

Bottlenecks:

  level | a. intensity |   performance    |   bandwidth   | bandwidth kernel
--------+--------------+-----------------+--------------+------------------
    CPU |               |   21.60 GFLOP/s |               |
 CPU-L1 | 0.083 FLOP/b |    8.50 GFLOP/s |  102.01 GB/s | triad
  L1-L2 |   0.1 FLOP/b |    5.12 GFLOP/s |   51.15 GB/s | triad
  L2-L3 |   0.1 FLOP/b |    3.15 GFLOP/s |   31.48 GB/s | triad
 L3-MEM |  0.17 FLOP/b |    2.90 GFLOP/s |   17.40 GB/s | copy

Cache or mem bound
2.90 GFLOP/s due to L3-MEM transfer bottleneck (bw with from copy benchmark)
Arithmetic Intensity: 0.17 FLOP/b
```

Summary & remarks

- ECM can
  - predict single-core performance and scaling behavior of streaming kernels
  - predict the impact of intended optimizations and code changes
  - be more accurate than Roofline but (in principle) requires less input data

- ECM has problems with
  - reproducing scaling behavior near saturation
  - extremely tight kernels on fast memory interfaces (too optimistic)

- Possible refinement: latency penalties

ERLANGEN REGIONAL
COMPUTING CENTER

DFG Priority Programme1648

Bavarian Network for HPC

Thank You.

Holger Stengel
Julian Hammer
Jan Treibig
Gerhard Wellein