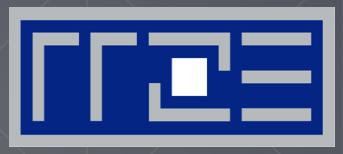
ERLANGEN REGIONAL COMPUTING CENTER



Performance engineering (of streaming loop kernels) via analytical models

Georg Hager
Erlangen Regional Computing Center (RRZE)

ISC15 Workshop “Performance Modeling: Methods & Applications”
July 16, 2015, Frankfurt, Germany

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.* Concurrency Computat.: Pract. Exper. (2015), DOI: 10.1002/cpe.3489
- H. Stengel, J. Treibig, G. Hager, and G. Wellein: *Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model*. Proc. ICS'15, the 29th International Conference on Supercomputing, Newport Beach, CA, June 8-11, 2015. DOI: 10.1145/2751205.2751240

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

- J. Hofmann, D. Fey, J. Eitzinger, G. Hager, G. Wellein: *Performance analysis of the Kahan-enhanced scalar product on current multicore processors.* Accepted for PPAM2015. Preprint: arXiv:1505.02586

Motivation

Analytical performance modeling:

"Constructing a simplified model for the interaction between software and hardware in order to understand lowest-order performance behavior"

Basic questions addressed by analytic performance models

- What is the bottleneck? → optimization technique
- What is the next bottleneck? → performance potential of the optimization
- Impact of processor frequency and socket scalability
  → Appropriate execution parameters, energy-optimized operating point

If the model fails, we learn something!

Agenda

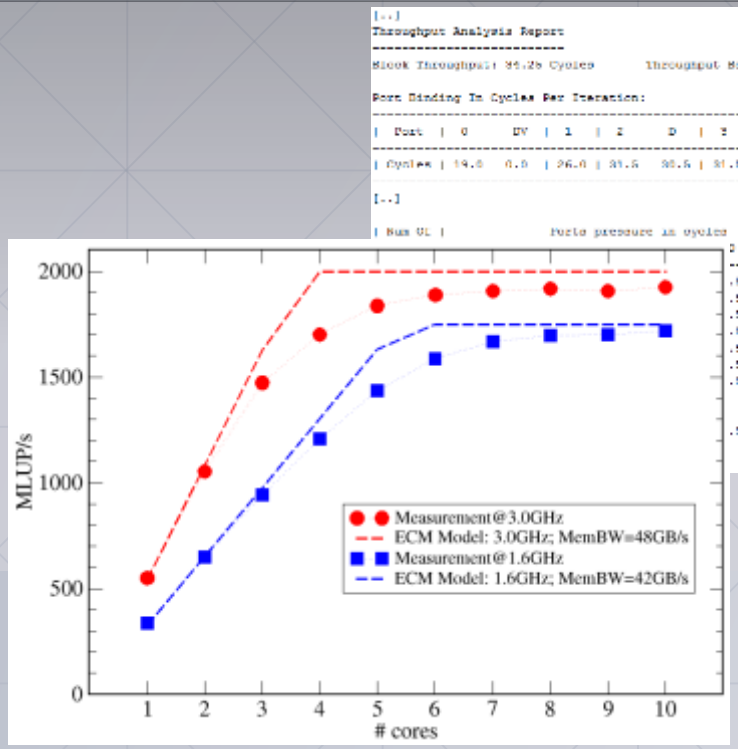
- ECM model
  - Basic rules, non-overlap
  - Notation
  - Saturation and comparison with Roofline

- Case study 1: 5-pt 2D stencil (“Jacobi”)
- Case study 2: UXX stencil (SP/DP)
- Case study 3: Kahan-enhanced scalar product

- Towards model automation: kerncraft

- Summary & outlook

THE ECM MODEL

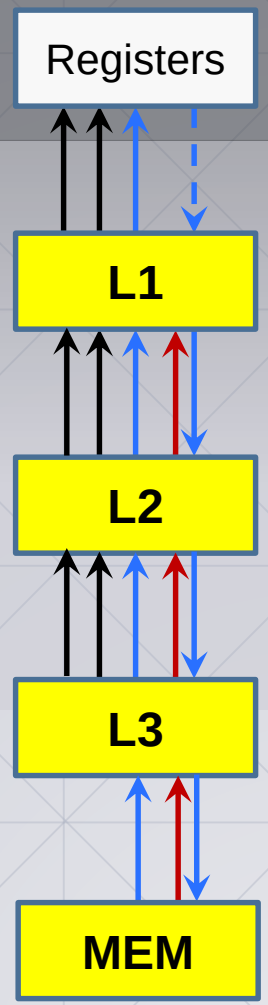



```
[..]
Throughput Analysis Report
--------------------------
Block Throughput: 36.2b Cycles       Throughput Bottleneck: FrontEnd

Port Binding In Cycles Per Iteration:
-------------------------------------------------------------------------
|  Port  |  0     DV  |  1   |  2     D   |  3     D   |  4   |  5   |
-------------------------------------------------------------------------
| Cycles | 14.0   0.0 | 26.0 | 36.5  36.5 | 36.5  36.5 | 2.0  | 24.0 |
-------------------------------------------------------------------------

[..]

| Num Of |              Ports pressure in cycles                |    |
|    0   |  4   |  5   |    |
---------------------
.5 |      |      |    | vmovupd xmm1, xmmword ptr [r12+r14*4+0x10]
.5 |      |      |    | vmovupd xmm0, xmmword ptr [rsi+r10*8+0x10]
.5 |      |      |    | vmovupd xmm13, xmmword ptr [r12+r14*4+0x14]
.5 |      |      |    | vmovupd xmm14, xmmword ptr [r12+r14*4+0xc]
.5 |      |      |    | vmovupd xmm12, xmmword ptr [rsi+r10*8+0x20]
.5 |      |      |    | mov r15, qword ptr [rsp+0x338]
.5 |      | 1.0  |    | vinsertf128 ymm10, ymm1, xmmword ptr [r12+r
   |      |      |    | vaddpd ymm1, ymm7, ymm10
   |      |      |    | vmulpd ymm15, ymm8, ymm10
.5 |      | 1.0  |    | vinsertf128 ymm11, ymm0, xmmword ptr [rsi+r
   |      |      |    | vaddpd ymm0, ymm1, ymm11
```



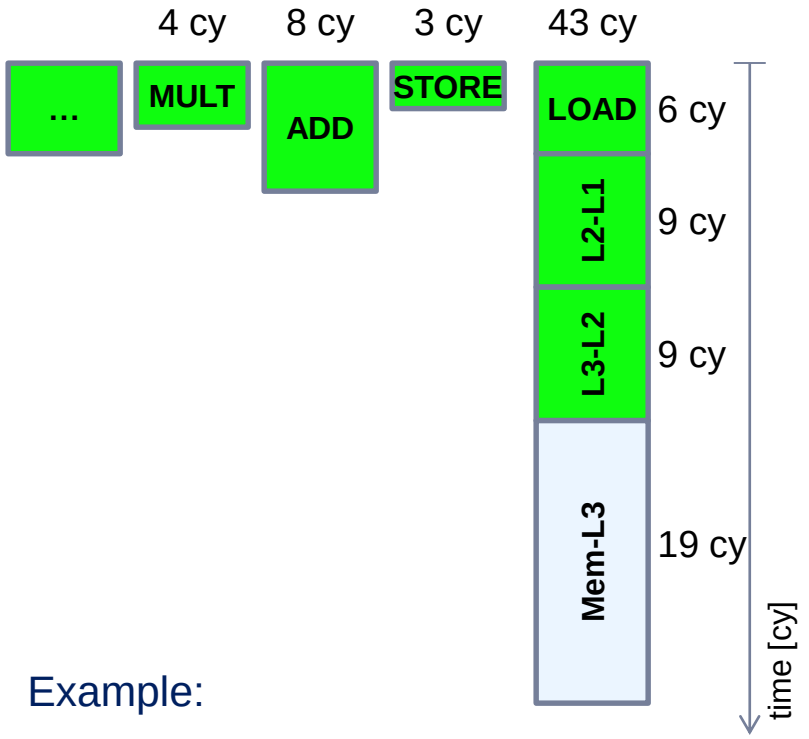
ECM model – predicting execution time for loop kernels

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 may 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

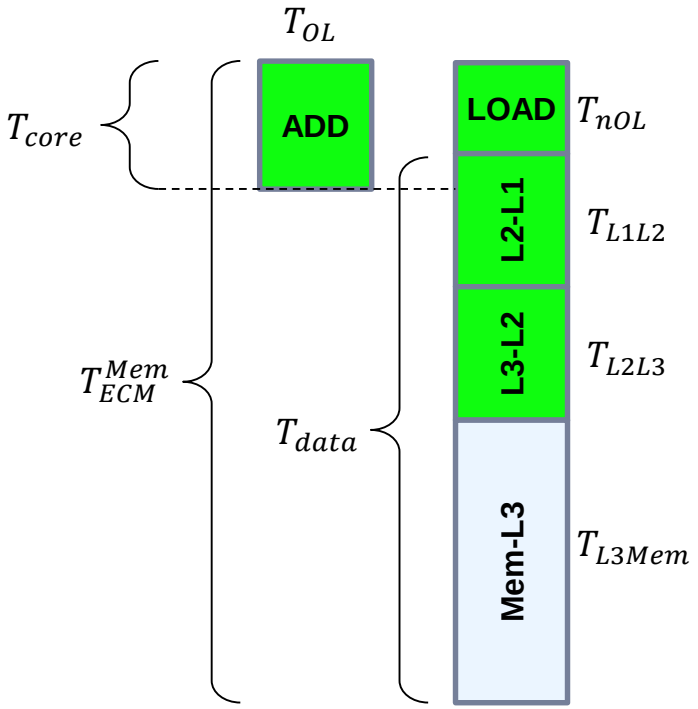
ECM model – composition

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:

$$\{\, T_{OL} \parallel T_{nOL} \mid T_{L1L2} \mid T_{L2L3} \mid T_{L3Mem} \,\}$$

cy invariant to clock speed

cy changes w/ clock speed

Example from previous slide:

$$\{\, 8 \parallel 6 \mid 9 \mid 9 \mid 19 \,\}\ \text{cy}$$

ECM model – prediction

Notation for cycle predictions in different memory hierarchy levels:

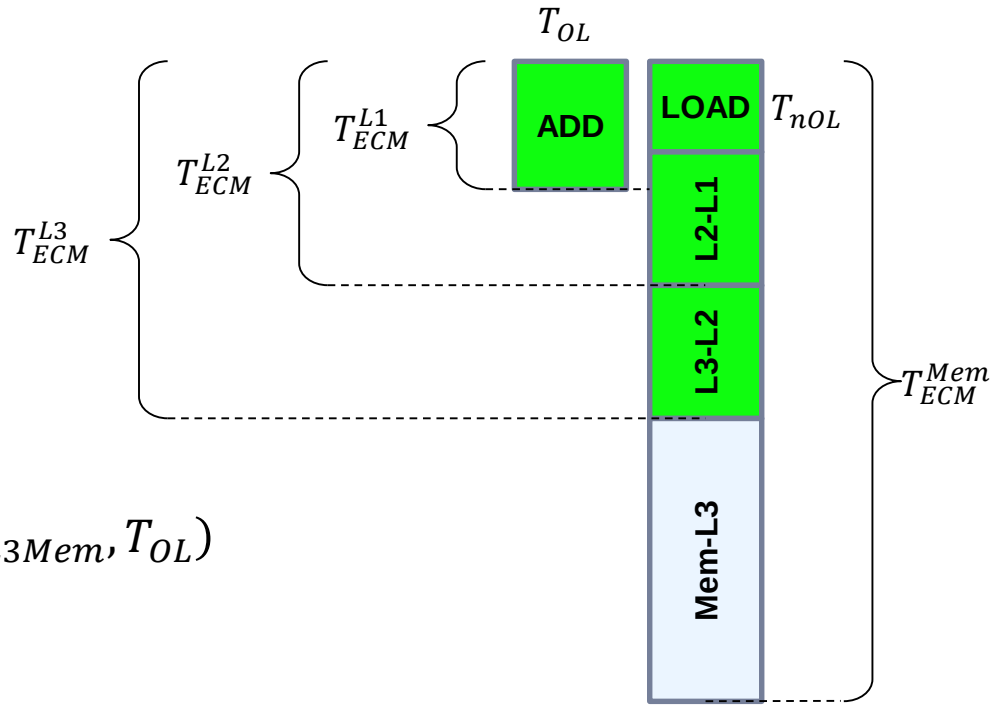
$$\{ T_{ECM}^{L1} \rceil T_{ECM}^{L2} \rceil T_{ECM}^{L3} \rceil T_{ECM}^{Mem} \}$$

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

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

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

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



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 – saturation

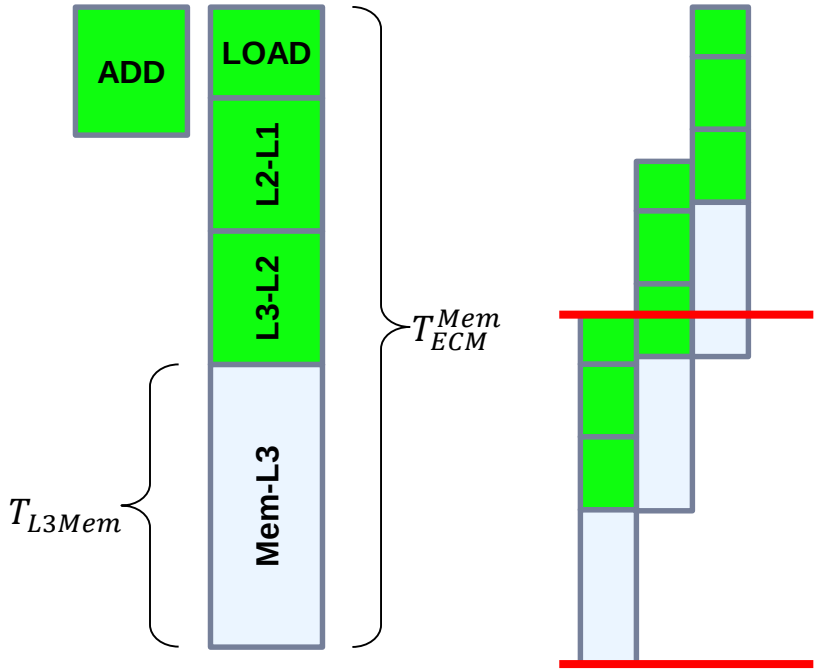
Main assumption: Performance scaling is linear until a bandwidth bottleneck (b_S) is hit

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:

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

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:

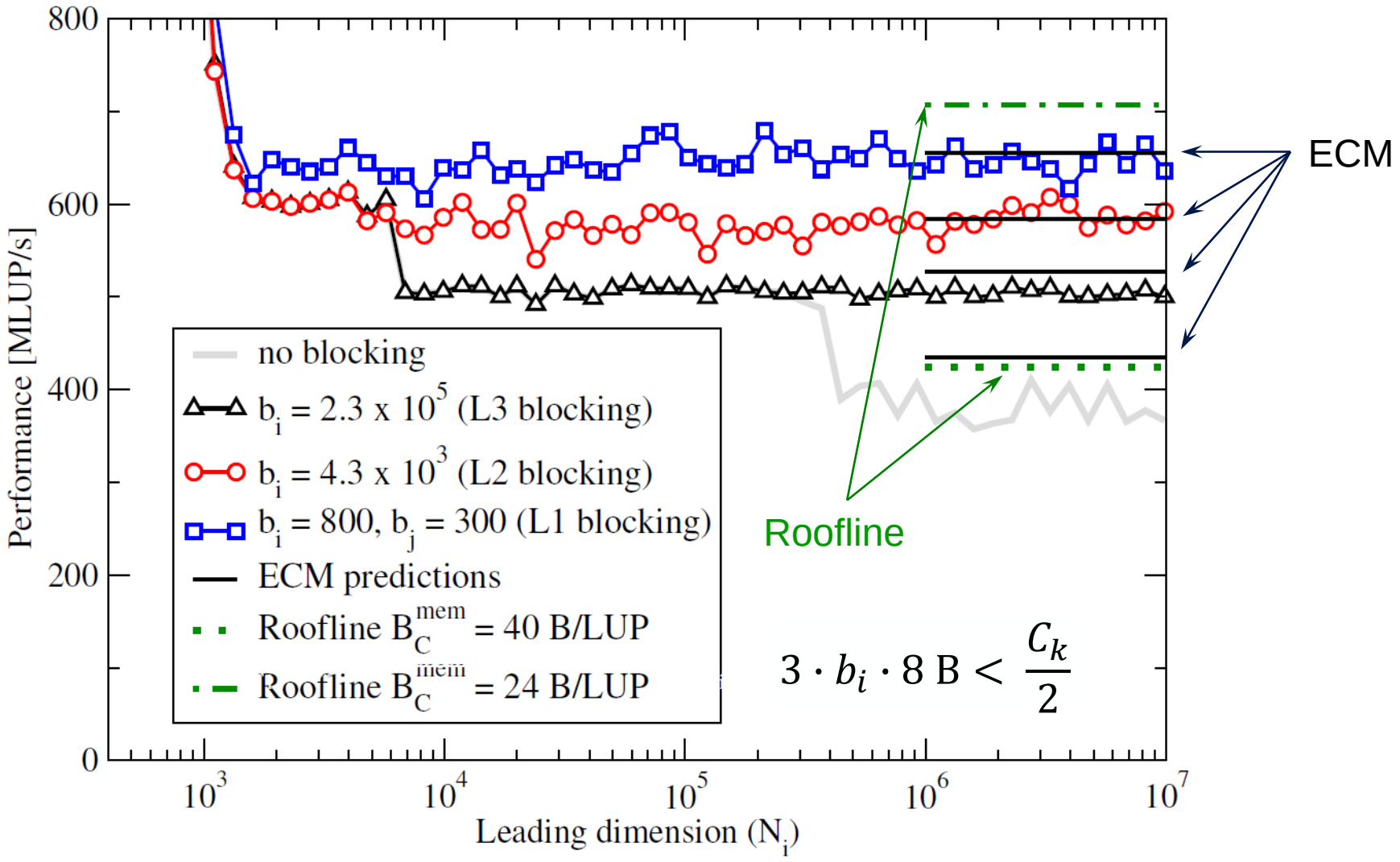
$$(2r+1) \cdot N_i \cdot 8\,\mathrm{B} < \frac{C_k}{2}$$

2D 5-pt: $r = 1$

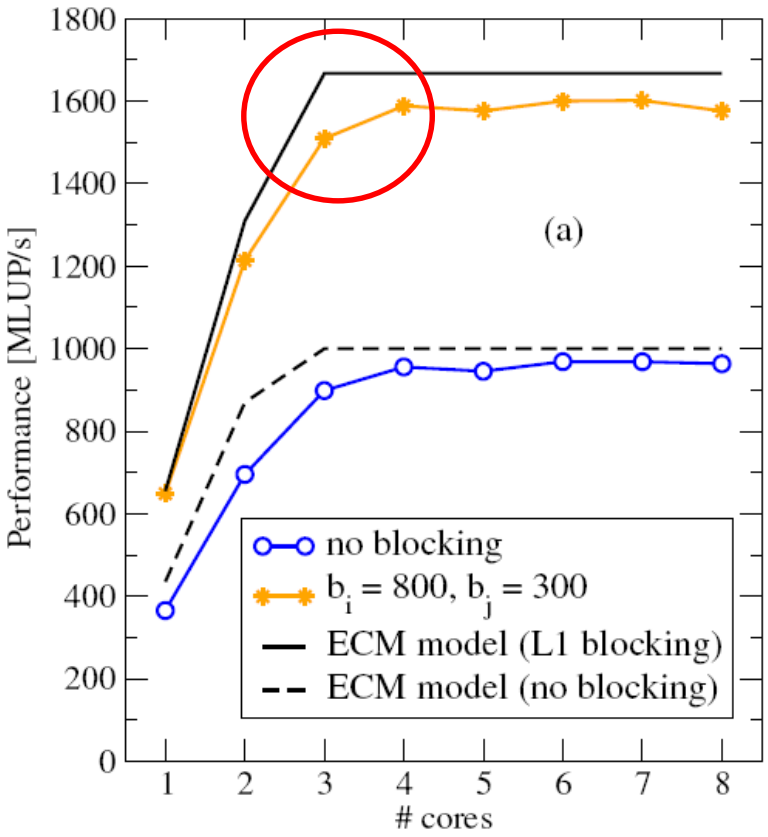
LC	ECM Model [cy]	prediction [cy]	$P^{\mathrm{mem}}_{\mathrm{ECM}}$ [MLUPS]	$N_i <$	n_S
L1	{6 ‖ 8 | 6 | 6 | 13}	{8 ⌉ 14 ⌉ 20 ⌉ 33}	659	683	3
L2	{6 ‖ 8 | 10 | 6 | 13}	{8 ⌉ 18 ⌉ 24 ⌉ 37}	587	5461	3
L3	{6 ‖ 8 | 10 | 10 | 13}	{8 ⌉ 18 ⌉ 28 ⌉ 41}	529	436900	4
—	{6 ‖ 8 | 10 | 10 | 22}	{8 ⌉ 18 ⌉ 28 ⌉ 50}	438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

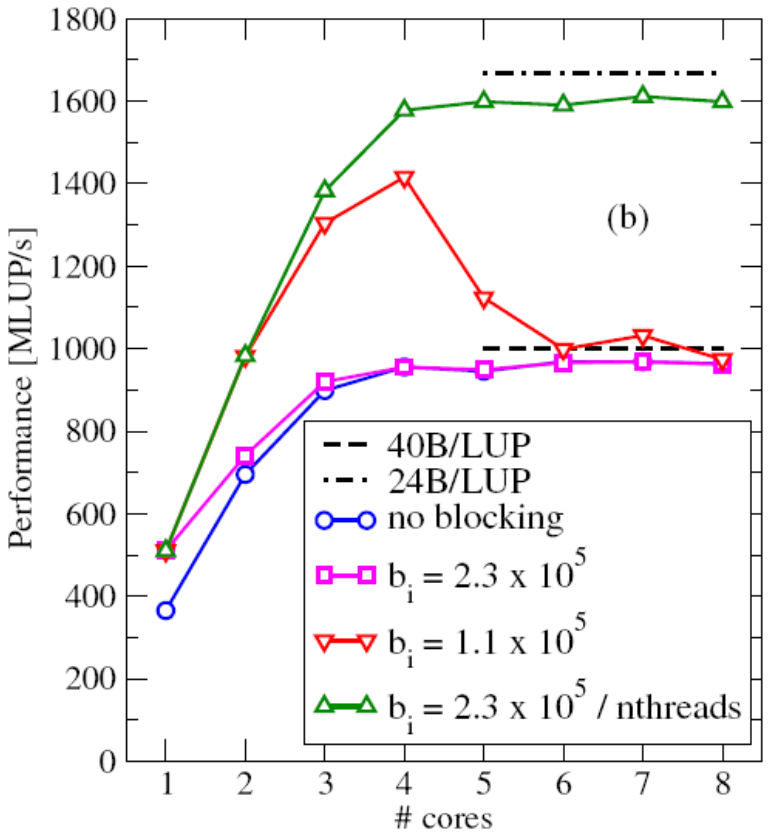
LC = layer condition satisfied in …

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



2D 5-pt multi-core scaling

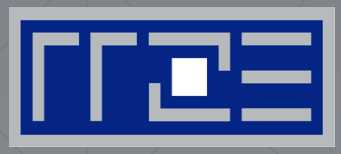




Modified layer condition for static work-sharing of outer loop:

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

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

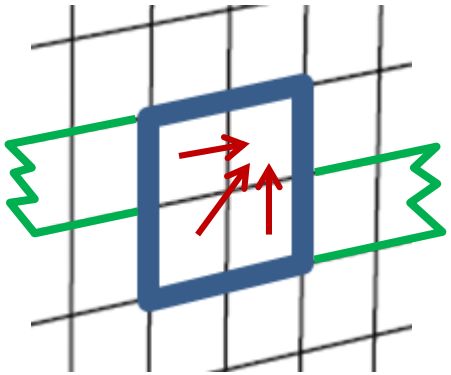


```
#pragma omp parallel for private(d) schedule(static)
for(int k=2; k<=N-1; k++){
  for(int j=2; j<=N-1; j++){
    for (int i=2; i<=N-1; i++){
      d = 0.25*(d1[ k ][j][i] + d1[ k ][j-1][i]
              + d1[k-1][j][i] + d1[k-1][j-1][i]);
      u1[k][j][i] = u1[k][j][i] + (dth/d)
            *( c1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

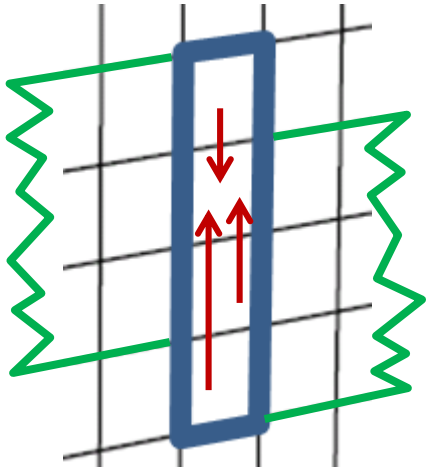
Should we try to eliminate the divide?

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 $N_i \times N_j \times N_k$):

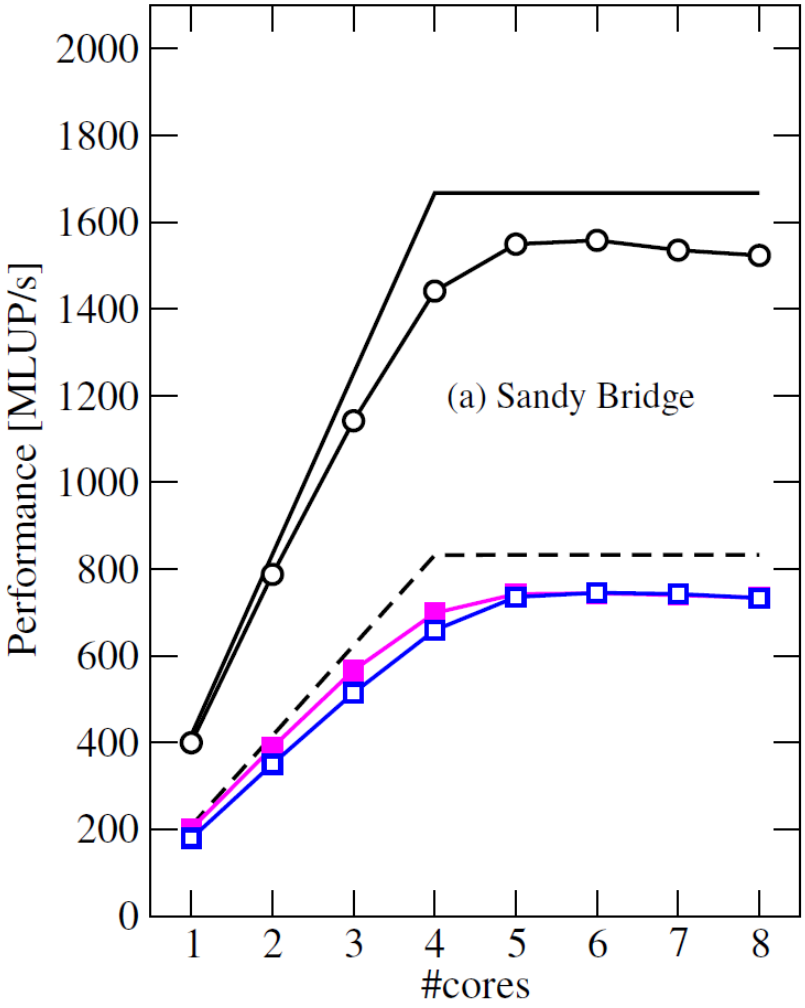
$$(4+2) \cdot N_i \cdot b_j \cdot n_{threads} \cdot \left\{ \begin{matrix} 4\text{ B (SP)} \\ 8\text{ B (DP)} \end{matrix} \right\} < \frac{C_3}{2}$$

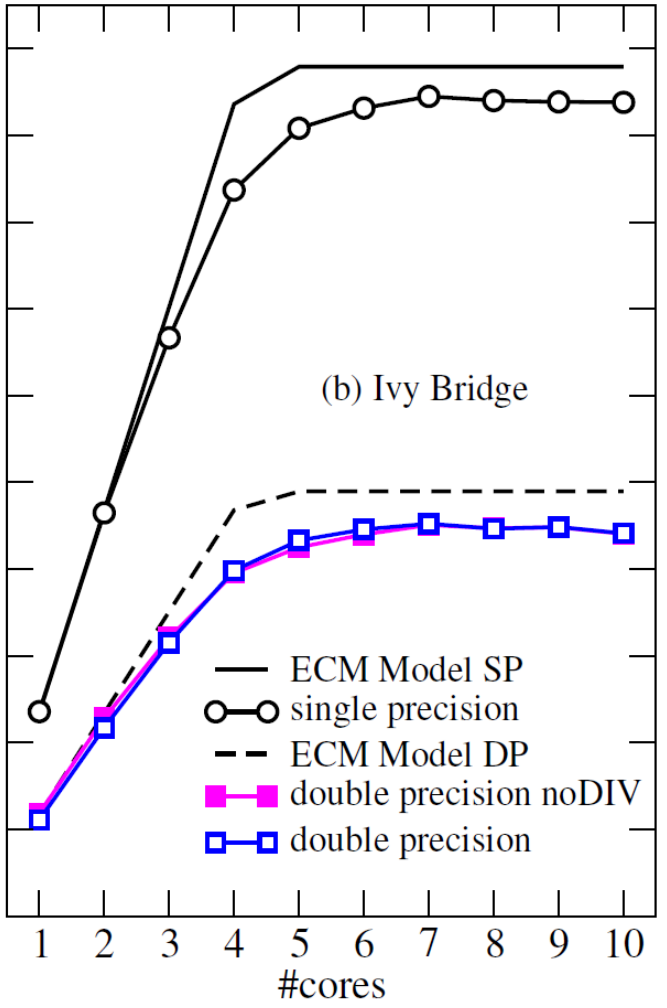
version	ECM model [cy]	prediction [cy]
DP	{84 ‖ 38 | 20 | 20 | 26}	{84 ⌉ 84 ⌉ 84 ⌉ 104}
SP	{45 ‖ 38 | 20 | 20 | 26}	{45 ⌉ 58 ⌉ 78 ⌉ 104}
DP noDIV	{41 ‖ 38 | 20 | 20 | 26}	{41 ⌉ 58 ⌉ 78 ⌉ 104}

Consequences:

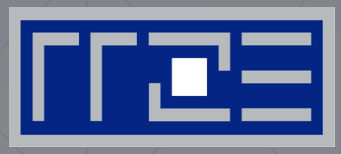
- No use in removing DIV from loop in DP for in-memory case
- No actual DIV in code for SP (compiler employs rcpps + NR)
- Temporal blocking not much use for serial, but makes the code scalable!

Uxx performance and scaling on SNB and IVB





KAHAN-ENHANCED SCALAR PRODUCT



Is the Kahan scalar product harmful for performance?

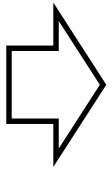
J. Hofmann, D. Fey, J. Eitzinger, G. Hager, G. Wellein: *Performance analysis of the Kahan-enhanced scalar product on current multicore processors.* Accepted for PPAM2015. Preprint: arXiv:1505.02586

Kahan-enhanced scalar product

```
float sum = 0.0;

for (int i=0; i<n; i++) {
   sum = sum + a[i] * b[i]
}
```

1 ADD, 1 MULT



```
float sum = 0.0;
float c = 0.0;
for (int i=0; i<N; ++i) {
    float prod = a[i]*b[i];
    float y = prod-c;
    float t = sum+y;
    c = (t-sum)-y;
    sum = t;
}
```

4 ADD, 1 MULT

- Does it harm performance to augment the dot kernel in this way?
- Is there are difference between single-threaded and multi-threaded?
- SP vs. DP? Influence of architecture?

ECM modeling of sdot on 10-core Ivy Bridge EP 2.2 GHz

Naive sdot (AVX):

$$\{2 \,\|\, 4 \,|\, 4 \,|\, 4 \,|\, 6.1 + 2.9\} \text{ cy}$$

Latency penalty

$$\{4 \rceil 8 \rceil 12 \rceil 18.1 + 2.9\} \text{ cy}$$

saturation at 4 cores

Kahan sdot, scalar mode:

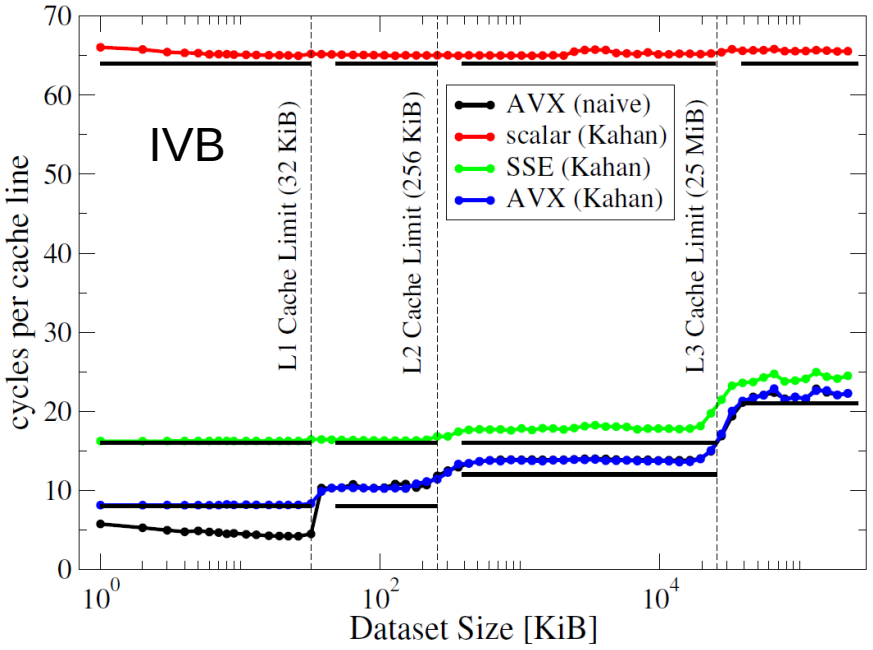
$$\{64 \,\|\, 16 \,|\, 4 \,|\, 4 \,|\, 6.1 + 2.9\} \text{ cy}$$
$$\{64 \rceil 64 \rceil 64 \rceil 64\} \text{ cy}$$

saturation at 11 cores

Comparing optimal AVX implementations on four Intel architectures (SP)

	ECM model [cy]	Prediction [cy/CL]	Pred. performance [GUP/s]
SNB	{8 ‖ 4 | 4 | 4 | 7.9 + 5.1}	{8 ⌉ 8 ⌉ 12 ⌉ 19.9 + 5.1}	{5.40 ⌉ 5.40 ⌉ 3.60 ⌉ 1.73}
IVB	{8 ‖ 4 | 4 | 4 | 6.1 + 2.9}	{8 ⌉ 8 ⌉ 12 ⌉ 18.1 + 2.9}	{4.40 ⌉ 4.40 ⌉ 2.93 ⌉ 1.68}
HSW	{8 ‖ **2** | **2** | 5.54 | 4.9 + 11.1}	{8 ⌉ 8 ⌉ 9.54 ⌉ 14.44 + 11.1}	{4.60 ⌉ 4.60 ⌉ 3.86 ⌉ 1.44}
BDW	{8 ‖ 2 | 2 | **4** | 7 + **1**}	{8 ⌉ 8 ⌉ 8 ⌉ 15 + 1}	{3.60 ⌉ 3.60 ⌉ 3.60 ⌉ 1.8}

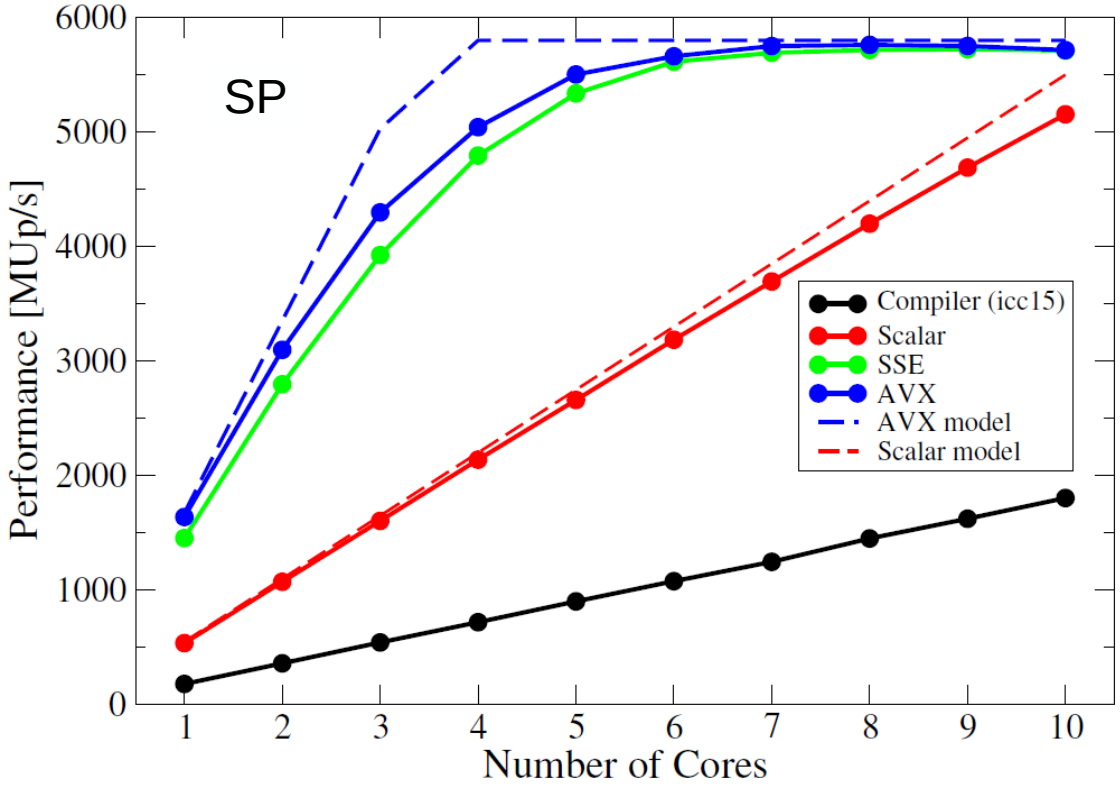
- Kahan without consequence if AVX is applied starting from L2 cache
- BDW latency panelty is rather low (== pure ECM model works well)



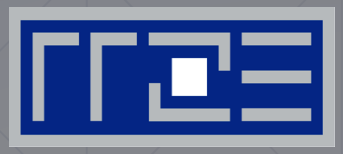
Saturation

- SP: Saturation possible if any kind of SIMD is applied
- DP: Saturates always

- Compiler is not able to generate decent code



KERNCRAFT



First steps towards automated model construction

kerncraft: ECM/Roofline modeling toolkit

GitHub, Inc. [US] https://github.com/cod3monk/kerncraft

GitHub This repository Search Explore Features Enterprise Blog

cod3monk / kerncraft Watch 1

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.

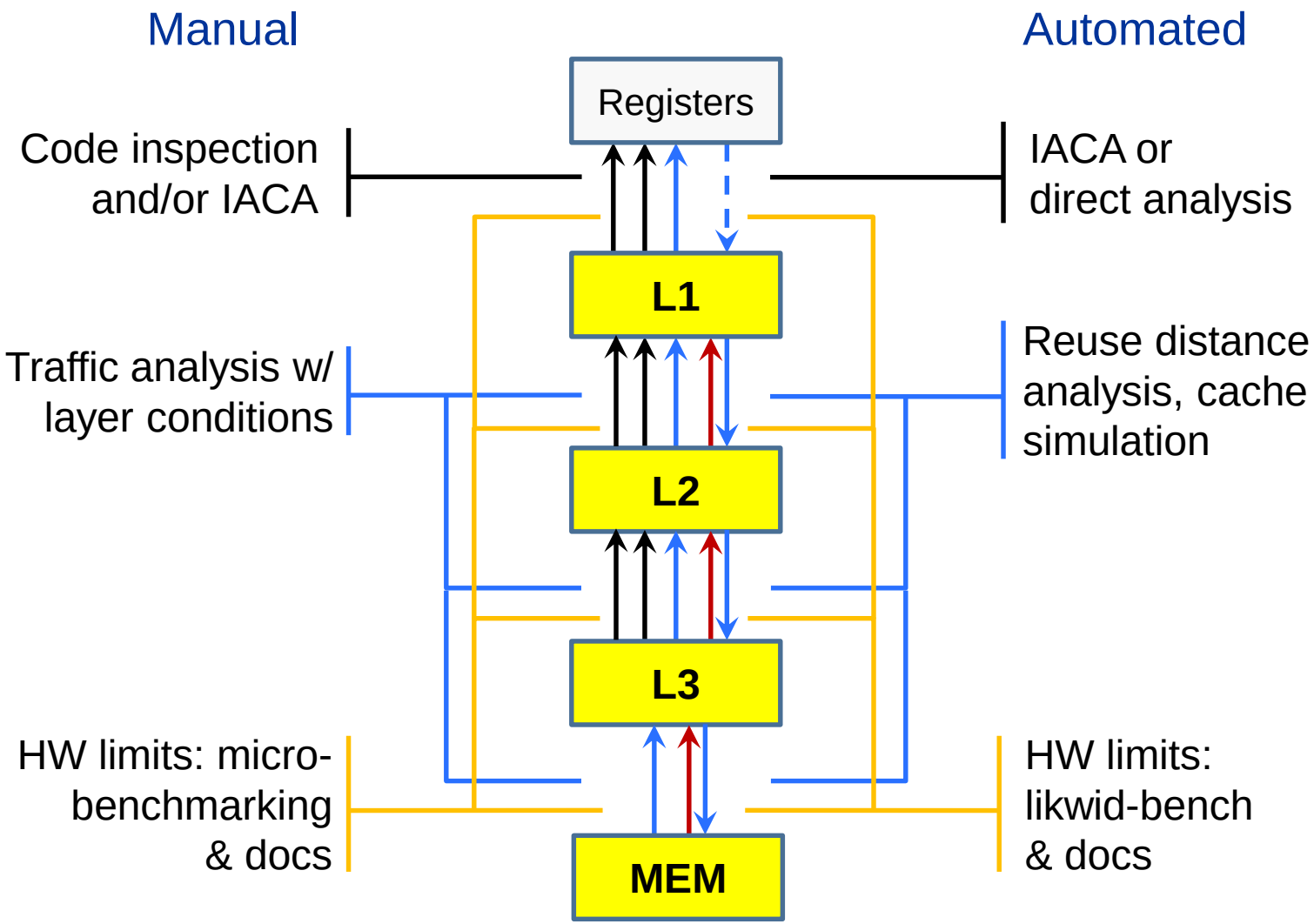
Installation

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



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
```

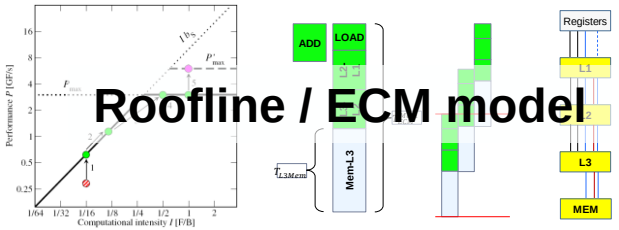
pycparser

IACA
TP/CP

AST

T_{OL}, T_{nOL}

Cache simulator/
reuse distance
analysis



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

```
[...]
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

[...]

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

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 & outlook

- 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 phenomenological input
  - be combined with a multicore power model for more insight:

M. Wittmann, G. Hager, T. Zeiser, J. Treibig, and G. Wellein: *Chip-level and multi-node analysis of energy-optimized lattice-Boltzmann CFD simulations.*
Concurrency Computat.: Pract. Exper. (2015), DOI: 10.1002/cpe.3489

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

- Approach to automating Roofline/ECM modeling:
  kerncraft https://github.com/cod3monk/kerncraft

ERLANGEN REGIONAL COMPUTING CENTER

DFG Priority Programme1648

Bavarian Network for HPC

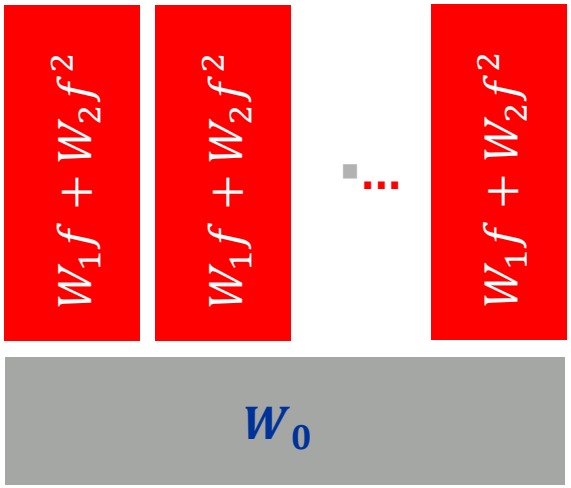
Thank You.

Holger Stengel
Johannes Hofmann
Julian Hammer
Jan Eitzinger
Gerhard Wellein

A simple power model for multicore chips

Model assumptions:

1. Power is a quadratic polynomial in the clock frequency: $W = W_0 + w_1 f + w_2 f^2$
2. Dynamic power is linear in the number of active cores: $W_{dyn} = (W_1 f + W_2 f^2) n$
3. Performance is linear in the number of cores until it hits a bottleneck
4. Performance is linear in the clock frequency unless it hits a bottleneck (simplification from performance models!)
5. Energy to solution is power dissipation divided by performance



Model:

$$E = \frac{\text{Power}}{\text{Performance}} = \frac{W_0 + (W_1 f + W_2 f^2) n}{P_{\text{ECM}}(n, f)}$$

Energy to solution vs. performance on the socket (SNB)