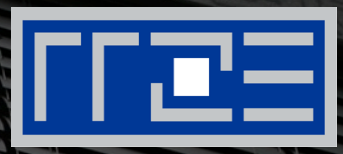
ERLANGEN REGIONAL
COMPUTING CENTER [RRZE]



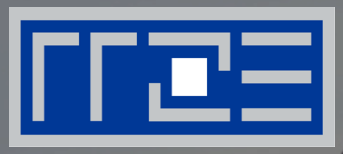
Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels

Julian Hammer
Georg Hager
Jan Eitzinger
Gerhard Wellein

Overview

1. Loop Kernels
2. Performance Modeling
    1. Roofline
    2. Execution Cache Memory
    3. Layer Conditions
3. Kerncraft
    1. Overview and Structure
    2. Output and Results
4. 3D-long-range Example

LOOP KERNELS



Kerncraft: A Tool for Analytic Performance Modeling of **Loop Kernels**

Loop Kernels

```
double a[5000], b[5000];
double s;

for(i=0; i<5000; ++i)
    a[i] = s * b[i];
```

```
double a[5000][5000];
double b[5000][5000];
double s;

for(j=1; j<5000-1; ++j)
    for(i=1; i<5000-1; ++i)
        b[j][i] = ( a[j][i-1] + a[j][i+1]
                  + a[j-1][i] + a[j+1][i] )
                  * s;
```

- Many inner-loop iterations
- No branching
- Access fully determined by loop counters (i.e., no irregularities)

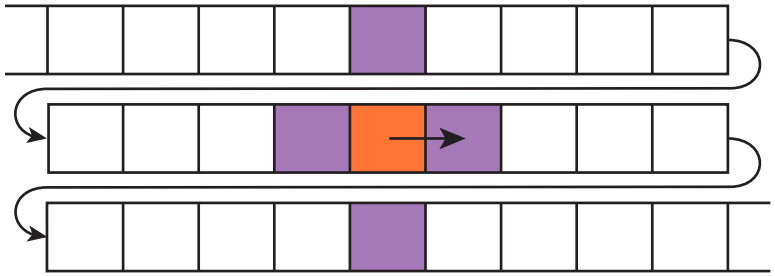
Loop Kernels

```
double a[5000], b[5000];
double s;

for(i=0; i<5000; ++i)
    a[i] = s * b[i];
```

Streaming Kernel

- Simple structure
- No data-reuse

```
double a[5000][5000];
double b[5000][5000];
double s;

for(j=1; j<5000-1; ++j)
    for(i=1; i<5000-1; ++i)
        b[j][i] = ( a[j][i-1] + a[j][i+1]
                  + a[j-1][i] + a[j+1][i] )
                  * s;
```



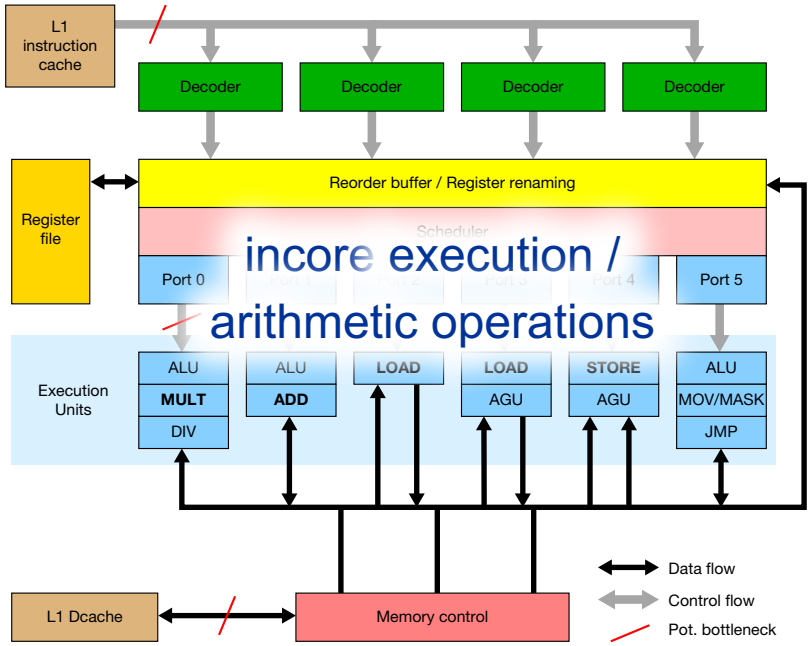
Stencil Code

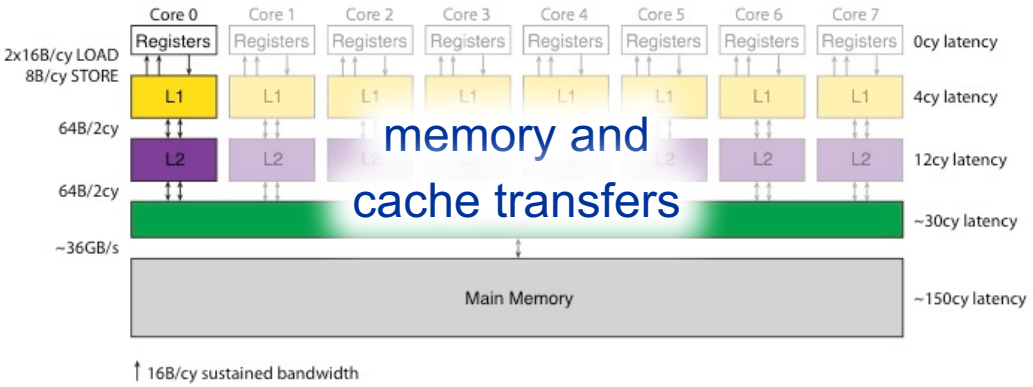
- Complex Structure
- Heavy data-reuse

Loop Kernels

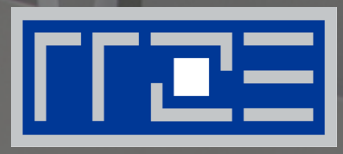
How to predict performance on complex architectures?

Two major contributions/bottlenecks:



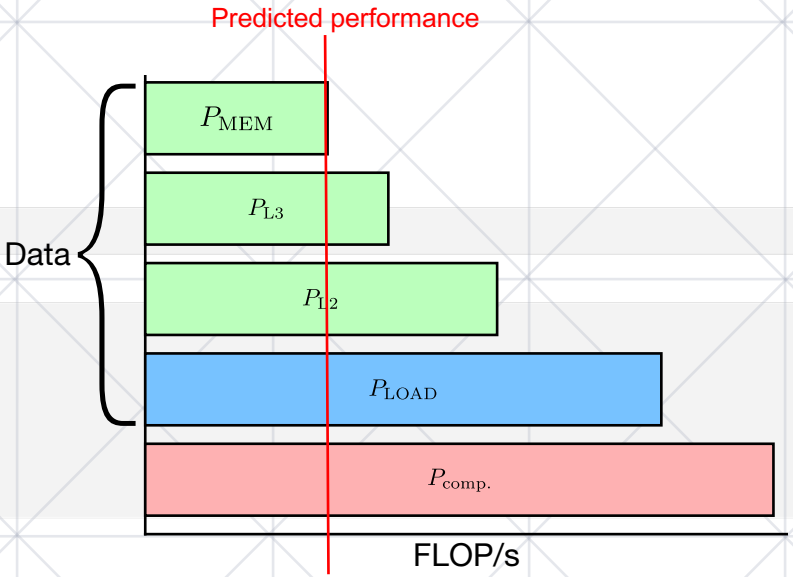


PERFORMANCE MODELING



Kerncraft: A Tool for **Analytic Performance Modeling** of Loop Kernels

Roofline



Roofline

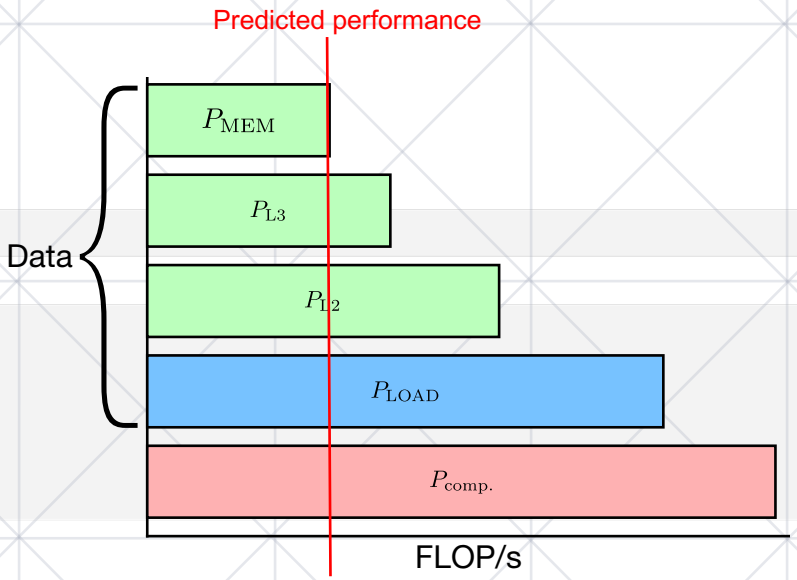
- All memory levels are separate bottlenecks

$$P = \min(P_{comp.}, I \bullet b_s)$$

$P_{comp.}$	Peak performance [FLOP/s]
I	Operational Intensity [FLOP/B]
b_s	Peak bandwidth [B/s]

- Bandwidths are measured by suitable benchmarks

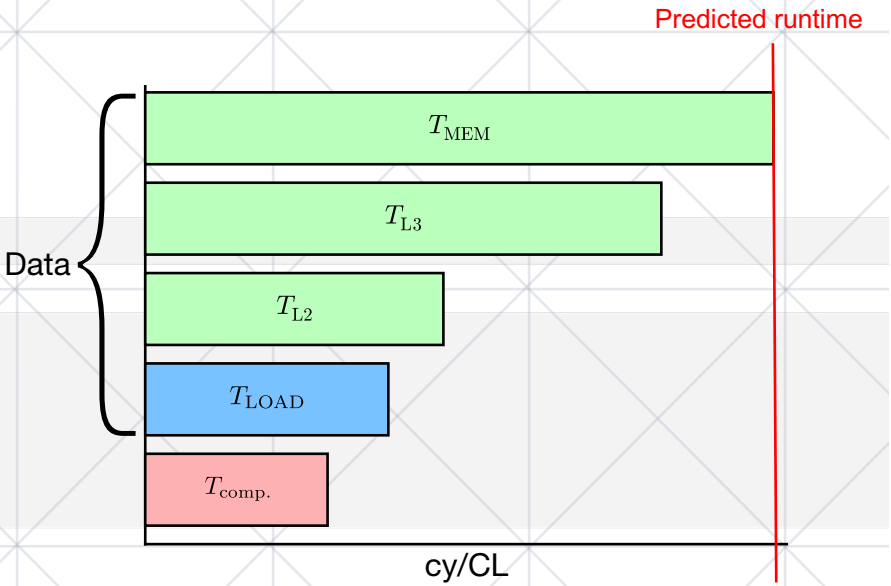
Roofline: Performance vs Time



Roofline

- CPU frequency?
  - → Cycles!
- Basic memory units?
  - → Cache Lines! (64 Byte)

Roofline: Performance vs Time



Roofline

- CPU frequency?
  - → Cycles!
- Basic memory units?
  - → Cache Lines! (1 CL=64 B)
  - → 1 unit of work = 1 CL

→ Cycle / Cache Line (cy/CL)

- Lower is better

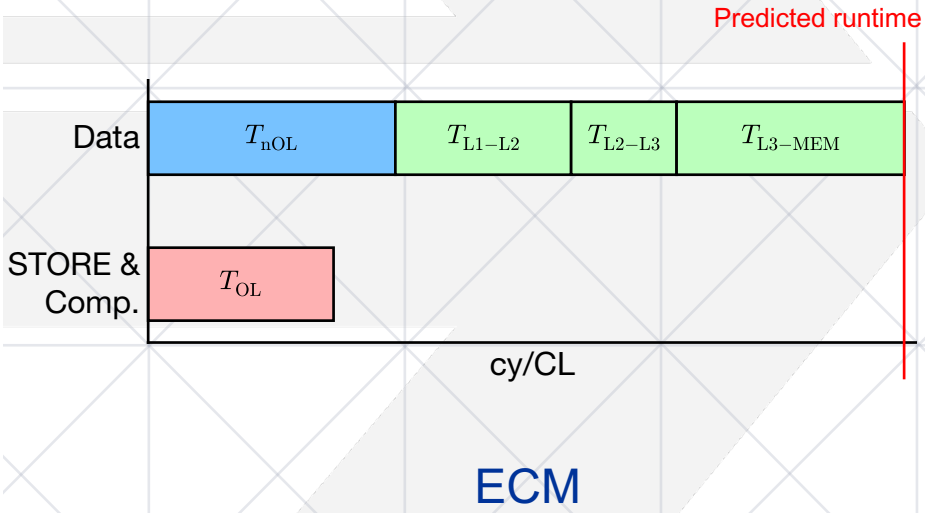
Execution-Cache-Memory (ECM) Model

- Memory and cache levels **contribute** to runtime

T_{OL} computation & stores
T_{nOL} loads from L1
$T_{L1\text{-}L2}$ loads from L2 into L1
$T_{L2\text{-}L3}$ loads from L3 into L2
$T_{L3\text{-}MEM}$ loads from main memory into L3

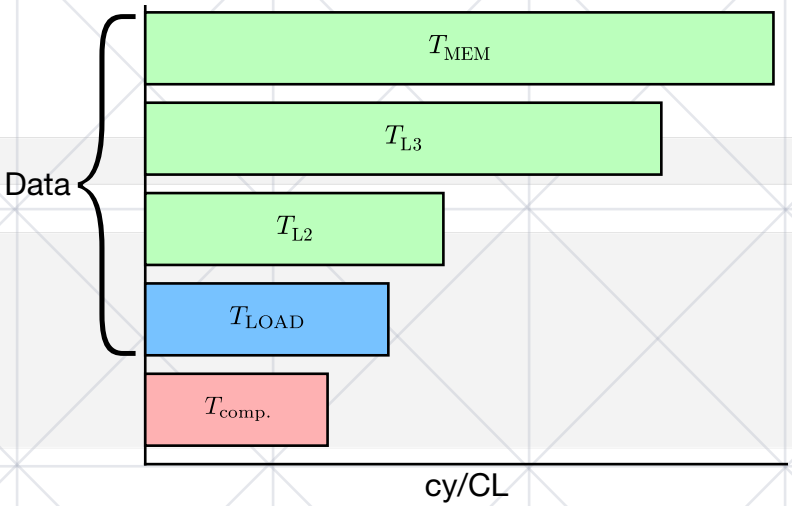
$\{ T_{OL} \,||\, T_{nOL} \mid T_{L1\text{-}L2} \mid T_{L2\text{-}L3} \mid T_{L3\text{-}MEM} \}$

- One measured input: full-socket mem. bandwidth

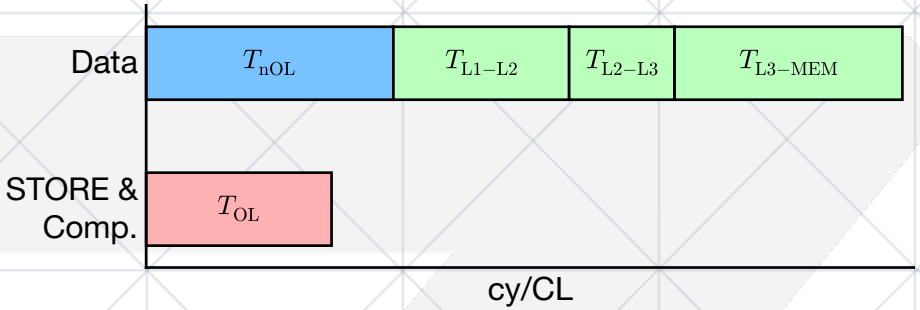


ECM

Roofline and ECM



Roofline



ECM

Performance Modeling

```
double a[5000], b[5000];
double s;

for(i=0; i<5000; ++i)
    a[i] = s * b[i];
```

STREAM Scale

- For each cache line (8 it.):
  - 1 CL is stored
  - 8 FLOP
  - 1 CL are loaded

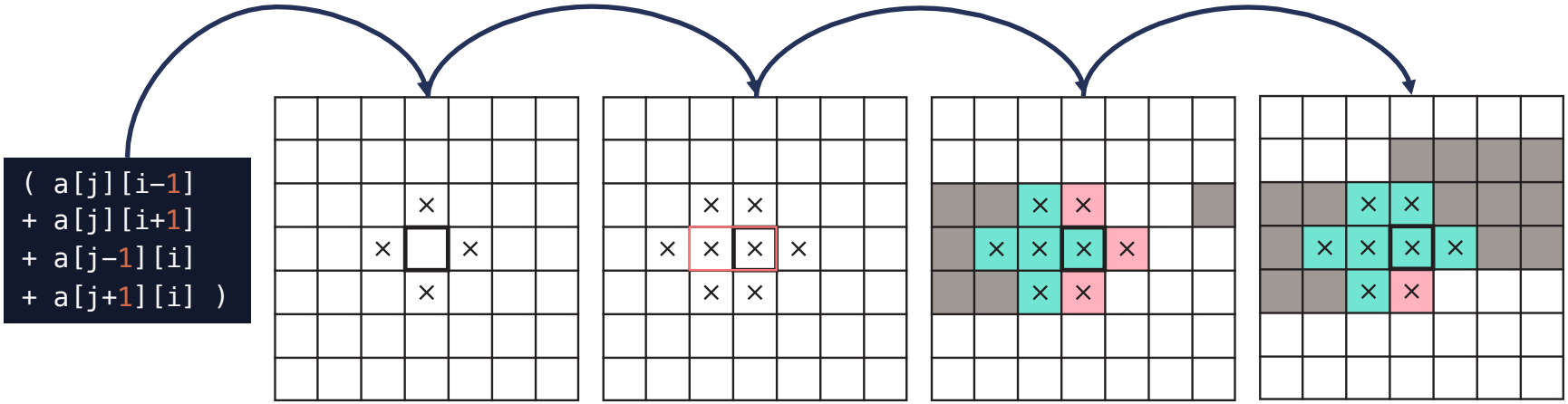
```
double a[5000][5000];
double b[5000][5000];
double s;

for(j=1; j<5000-1; ++j)
    for(i=1; i<5000-1; ++i)
        b[j][i] = ( a[j][i-1] + a[j][i+1]
                  + a[j-1][i] + a[j+1][i] )
                  * s;
```

2D 5-point Stencil

- For each cache line (8 it.):
  - 1 CL is stored
  - 32 FLOP
  - Up to 3 CL are loaded

Up to 3?

Layer Conditions



1D layer condition: stencil-width * stencil-height < cache-size

2D layer condition: stencil-height * matrix-width < cache-size

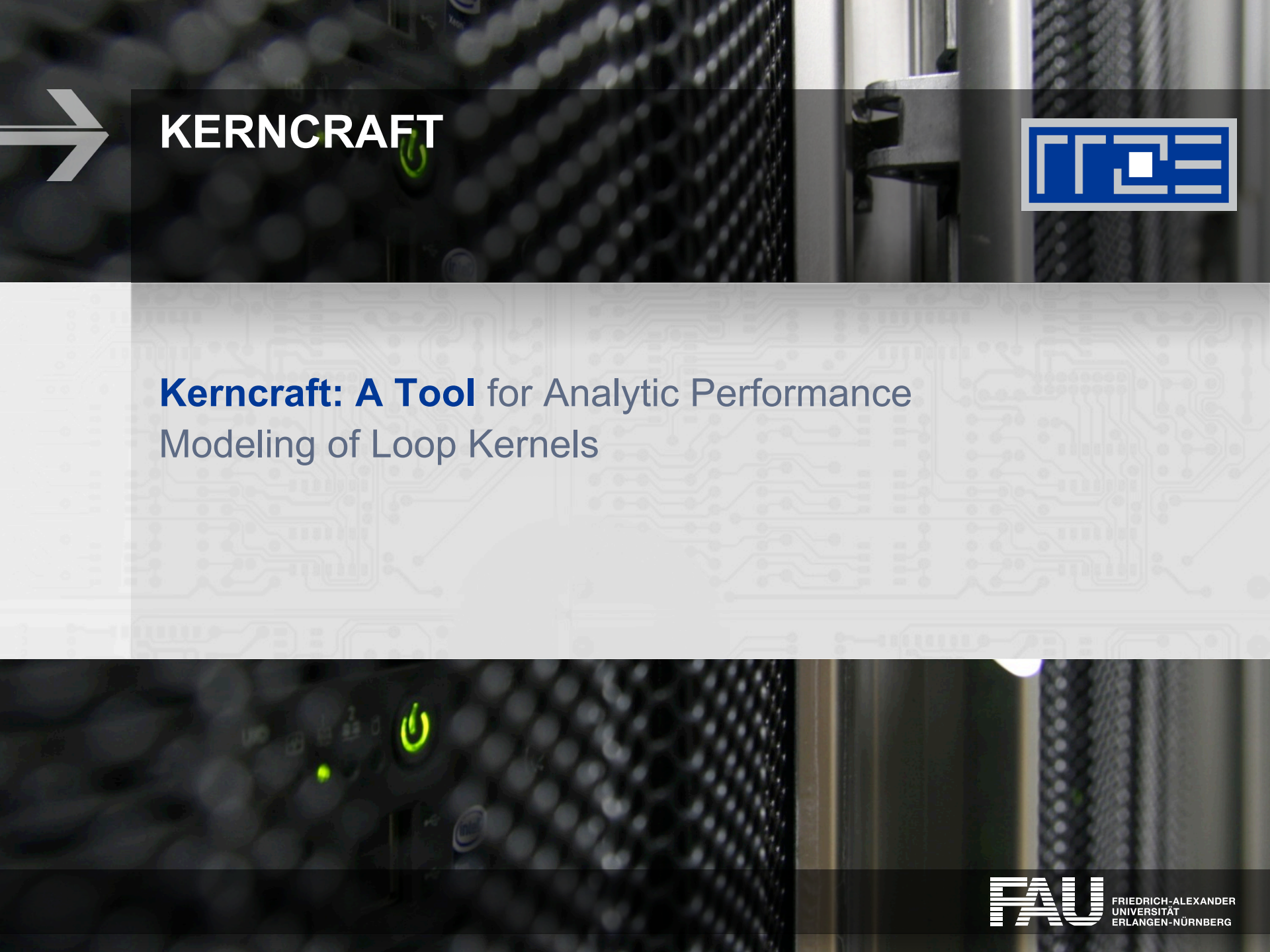
nD layer condition: $C_{\text{req.}} = \left(\sum L_{\text{rel.offsets}} + \max(L_{\text{rel.offsets}}) * n_{\text{slices}} \right) * s$

Potential Benefits

- Guided optimizations
  - Which optimization strategy to pursue?
  - What gain is to be expected?
- Guiding decisions for or against a specific architecture
  - What can be expected on new hardware?
  - Enabling hardware/software co-design
- Energy optimized computing
  - Which configuration yields the best energy to solution?
- Deeper understanding of code and hardware interactions

Performance Modeling

```
double U[M][N][N];
double V[M][N][N];
double ROC[M][N][N];
double c0, c1, c2, c3, c4, lap;

for(int k=4; k < M-4; k++) {
    for(int j=4; j < N-4; j++) {
        for(int i=4; i < N-4; i++) {
            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 ])
                + c2 * ( V[ k ][ j ][i+2] + V[ k ][ j ][i-2])
                + c2 * ( V[ k ][j+2][ i ] + V[ k ][j-2][ i ])
                + c2 * ( V[k+2][ j ][ i ] + V[k-2][ j ][ i ])
                + c3 * ( V[ k ][ j ][i+3] + V[ k ][ j ][i-3])
                + c3 * ( V[ k ][j+3][ i ] + V[ k ][j-3][ i ])
                + c3 * ( V[k+3][ j ][ i ] + V[k-3][ 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;
}}}
```

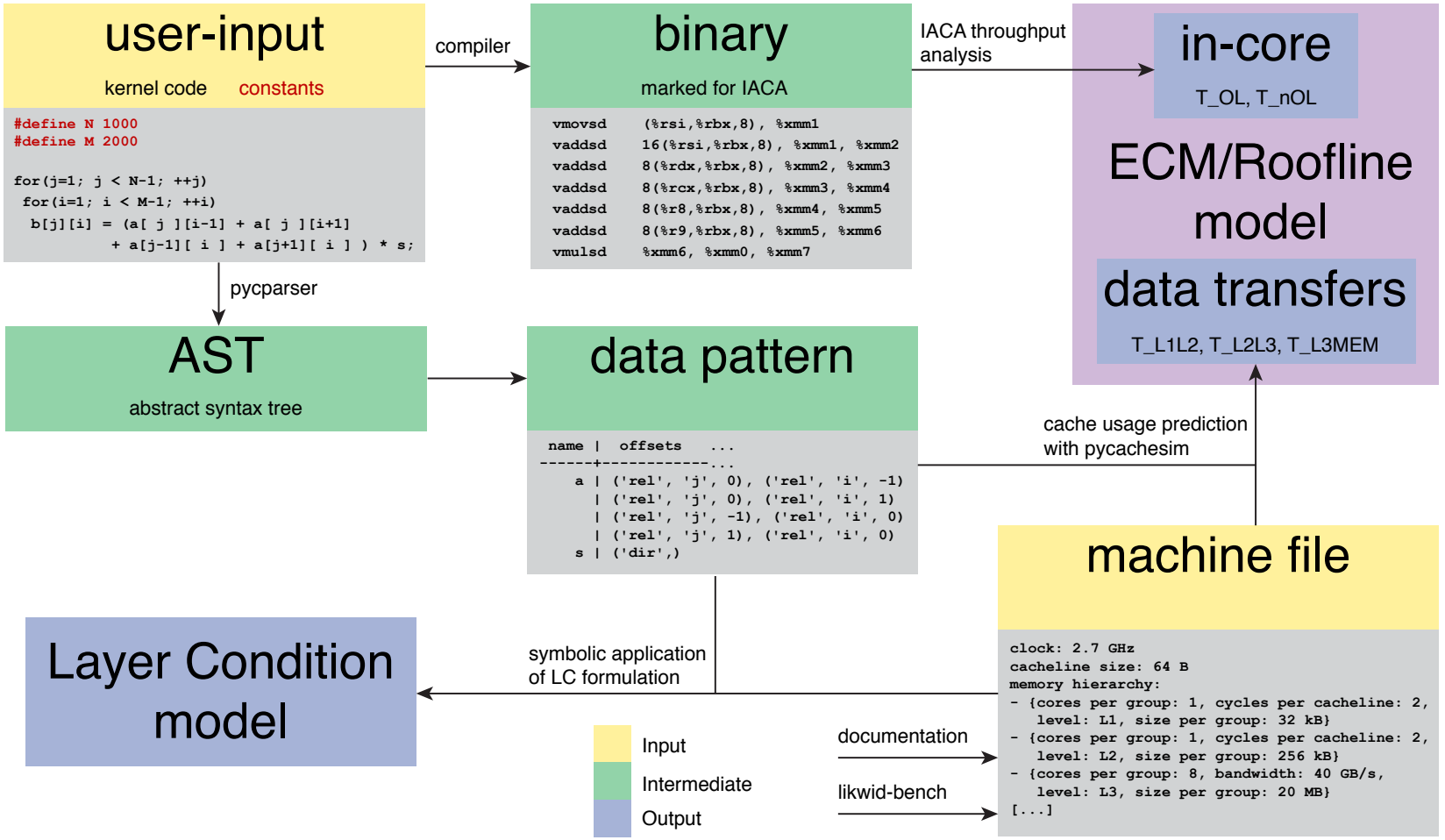


KERNCRAFT

Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Kerncraft


user-input
kernel code constants
#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
binary
marked for IACA
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
IACA throughput analysis
in-core
T_OL, T_nOL
ECM/Roofline model
data transfers
T_L1L2, T_L2L3, T_L3MEM
pycparser
AST
abstract syntax tree
data pattern
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',)
cache usage prediction with pycachesim
machine file
clock: 2.7 GHz
cacheline size: 64 B
memory hierarchy:
- {cores per group: 1, cycles per cacheline: 2, level: L1, size per group: 32 kB}
- {cores per group: 1, cycles per cacheline: 2, level: L2, size per group: 256 kB}
- {cores per group: 8, bandwidth: 40 GB/s, level: L3, size per group: 20 MB}
[...]
Layer Condition model
symbolic application of LC formulation
Input
Intermediate
Output
documentation
likwid-bench

Machine File

```
model type: Intel Core SandyBridge EP processor
model name: Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
clock: 2.7 GHz
sockets: 2
cores per socket: 8
threads per core: 2
cacheline size: 64 B
micro-architecture: SNB
FLOPs per cycle: {SP: {total: 16, ADD: 8, MUL: 8}
                  DP: {total: 8, ADD: 4, MUL: 4}
overlapping ports: ["0", "0DV", "1", "2", "3", "4", "5"]
non-overlapping ports: ["2D", "3D"]
compiler: icc
compiler flags: [-O3, -xAVX, -fno-alias]

memory hierarchy: […]

benchmarks:
  kernels: […]
  measurements: […]
```

Machine File – Memory Hierarchy

```
[…]
memory hierarchy:
     - level: L1
       cache per group: {'sets': 64, 'ways': 8, 'cl_size': 64, # 32 kB
                         'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True,
                         'load_from': 'L2', 'store_to': 'L2'}
       cores per group: 1
       threads per group: 2
       groups: 16
       cycles per cacheline transfer: 2
     - level: L2
       cache per group: {'sets': 512, 'ways': 8, 'cl_size': 64, # 256 kB
                         'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True,
                         'load_from': 'L3', 'store_to': 'L3'}
       cores per group: 1
       threads per group: 2
       groups: 16
       cycles per cacheline transfer: 2
     - level: L3
       cache per group: {'sets': 20480, 'ways': 16, 'cl_size': 64, # 20 MB
                         'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True}
       cores per group: 8
       threads per group: 16
       groups: 2
       cycles per cacheline transfer: null
     - level: MEM
       cores per group: 8
       threads per group: 16
[…]
```

Machine File – Benchmark Infos

```
[…]
benchmarks:
  kernels:
    copy:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 8.00 B, streams: 1}
    daxpy:
      FLOPs per iteration: 2
      read streams: {bytes: 16.00 B, streams: 2}
      read+write streams: {bytes: 8.00 B, streams: 1}
      write streams: {bytes: 8.00 B, streams: 1}
    load:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 0.00 B, streams: 0}
    triad:
      FLOPs per iteration: 2
      read streams: {bytes: 24.00 B, streams: 3}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 8.00 B, streams: 1}
    update:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 8.00 B, streams: 1}
      write streams: {bytes: 8.00 B, streams: 1}
  measurements: […]
```

Machine File – Benchmark Results

```
[…]
benchmarks:
  kernels: […]
  measurements:
    L1:
      1:
        cores: [1, 2, 3, 4, 5, 6, 7, 8]
        results:
          copy: [81.98 GB/s, 163.75 GB/s, 245.62 GB/s, 327.69 GB/s, 409.41 GB/s,
                 489.83 GB/s, 571.67 GB/s, 653.50 GB/s]
          daxpy: [71.55 GB/s, 143.01 GB/s, 214.86 GB/s, 286.26 GB/s, 355.60 GB/s,
                  426.71 GB/s, 497.45 GB/s, 568.97 GB/s]
[…]
        size per core: [16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB,
                        16.00 kB, 16.00 kB]
        size per thread: [16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB,
                          16.00 kB, 16.00 kB]
        threads: [1, 2, 3, 4, 5, 6, 7, 8]
        threads per core: 1
        total size: [16.00 kB, 32.00 kB, 48.00 kB, 64.00 kB, 80.00 kB, 96.00 kB,
                     112.00 kB, 128.00 kB]
[…]
    MEM:
      1:
        cores: [1, 2, 3, 4, 5, 6, 7, 8]
        results:
          copy: [11.60 GB/s, 21.29 GB/s, 25.94 GB/s, 27.28 GB/s, 27.47 GB/s, 27.36
              GB/s, 27.21 GB/s, 27.12 GB/s
[…]
```

Kerncraft – Output

ECM model: { T_{OL} || T_{nOL} | T_{L1-L2} | T_{L2-L3} | T_{L3-MEM} }

```
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel ECM -D N 5000 -D M 500
================================================================================
kernels/2d-5pt.c
================================================================================
{ 9.0 || 8.0 | 10
{ 9.0 \ 18.00 \ 2
saturating at 3 c
$
$ kerncraft --mac                                          cy/CL -D N 5000 -D M 500
================================================================================
kernels/2d-5pt.c
================================================================================
Cache or mem boun
29.79 cy/CL due to L3-MEM transfer bottleneck (bw from copy benchmark)
Arithmetic Intensity: 0.17 FLOP/b
$
```

```
double a[M][N];
double b[M][N];
double s;

for(j=1; j<M-1; ++j)
    for(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;
```

Kerncraft – Verbose Output

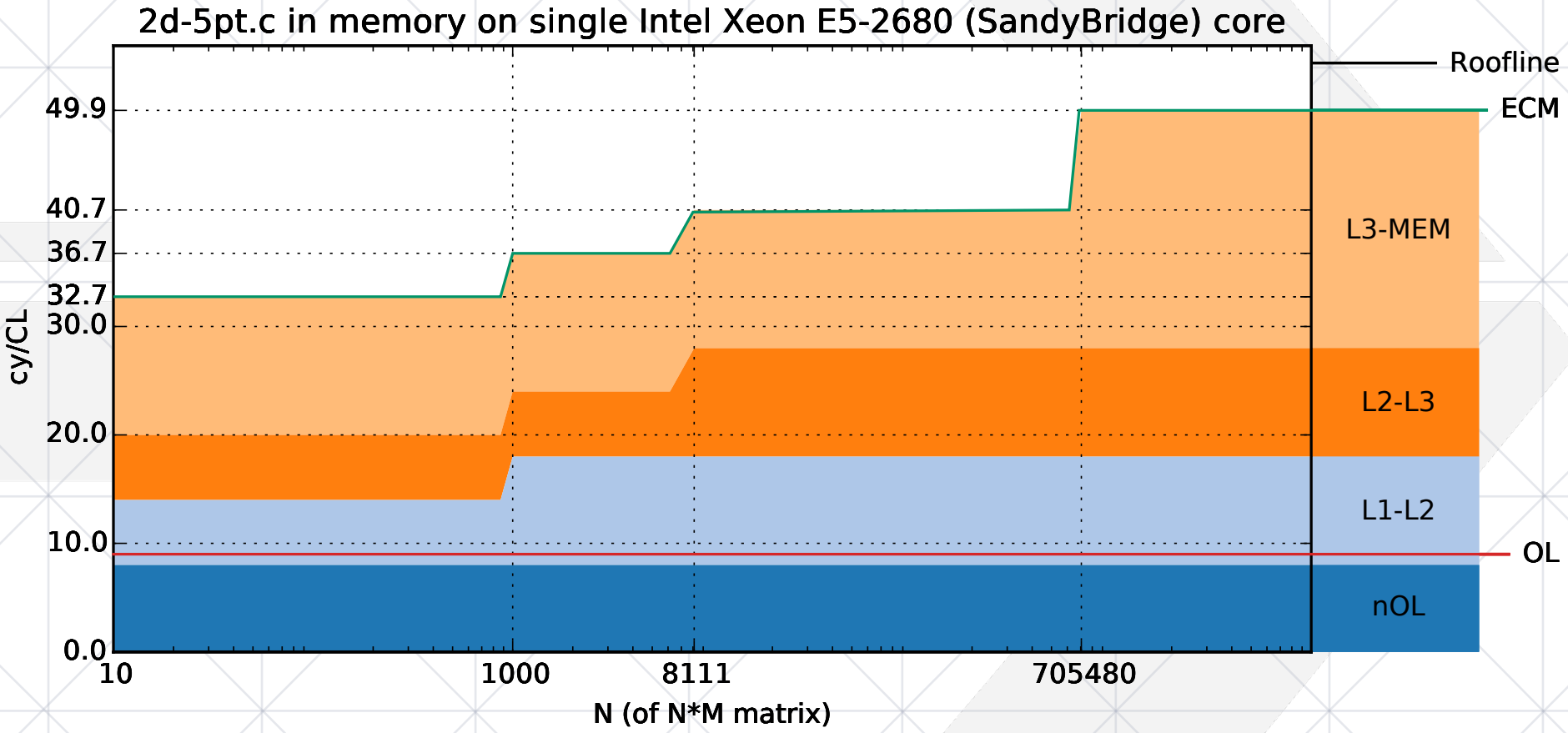
IACA analysis (for 2 CL, due to unrolling):

```
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel ECM -D N 5000 -D M 500 -vvv
[…]
Throughput Analysis Report
--------------------------
Block Throughput: 18.90 Cycles       Throughput Bottleneck: FrontEnd, PORT2_AGU, PORT3_AGUPort
Binding In Cycles Per Iteration:
-------------------------------------------------------------------------
|  Port  |  0   -  DV  |  1   |  2   -  D   |  3   -  D   |  4   |  5   |
-------------------------------------------------------------------------
| Cycles | 10.1  0.0  | 12.0 | 18.0  16.0 | 18.0  16.0 |  8.0 | 11.9 |
-------------------------------------------------------------------------
[…]
$
```

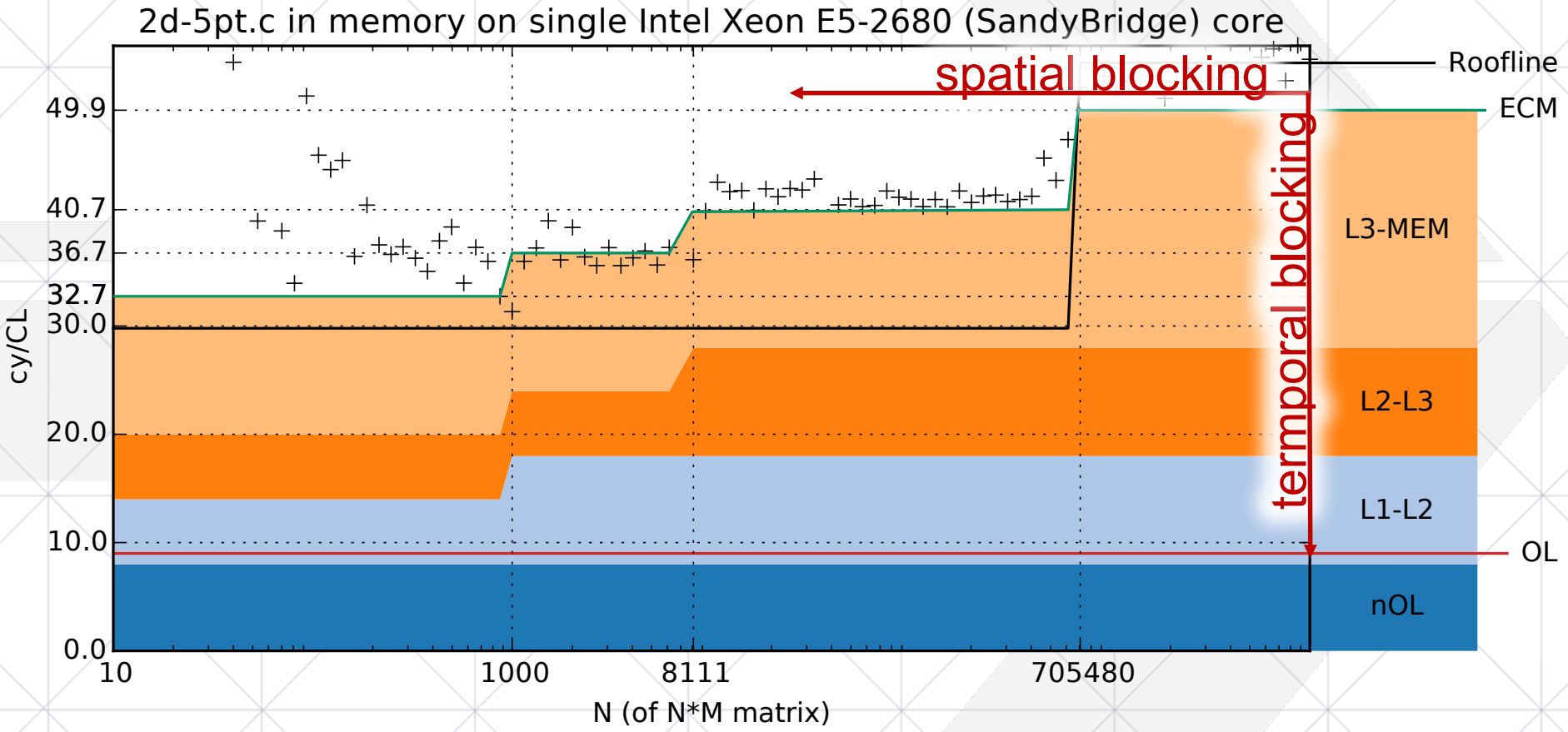
T_{OL}: max(port_cy) / unrolling = 18 cy / 2 CL = 9 cy/CL

T_{nOL}: max(data_port_cy) / unrolling = 16 cy / 2 CL = 8 cy/CL

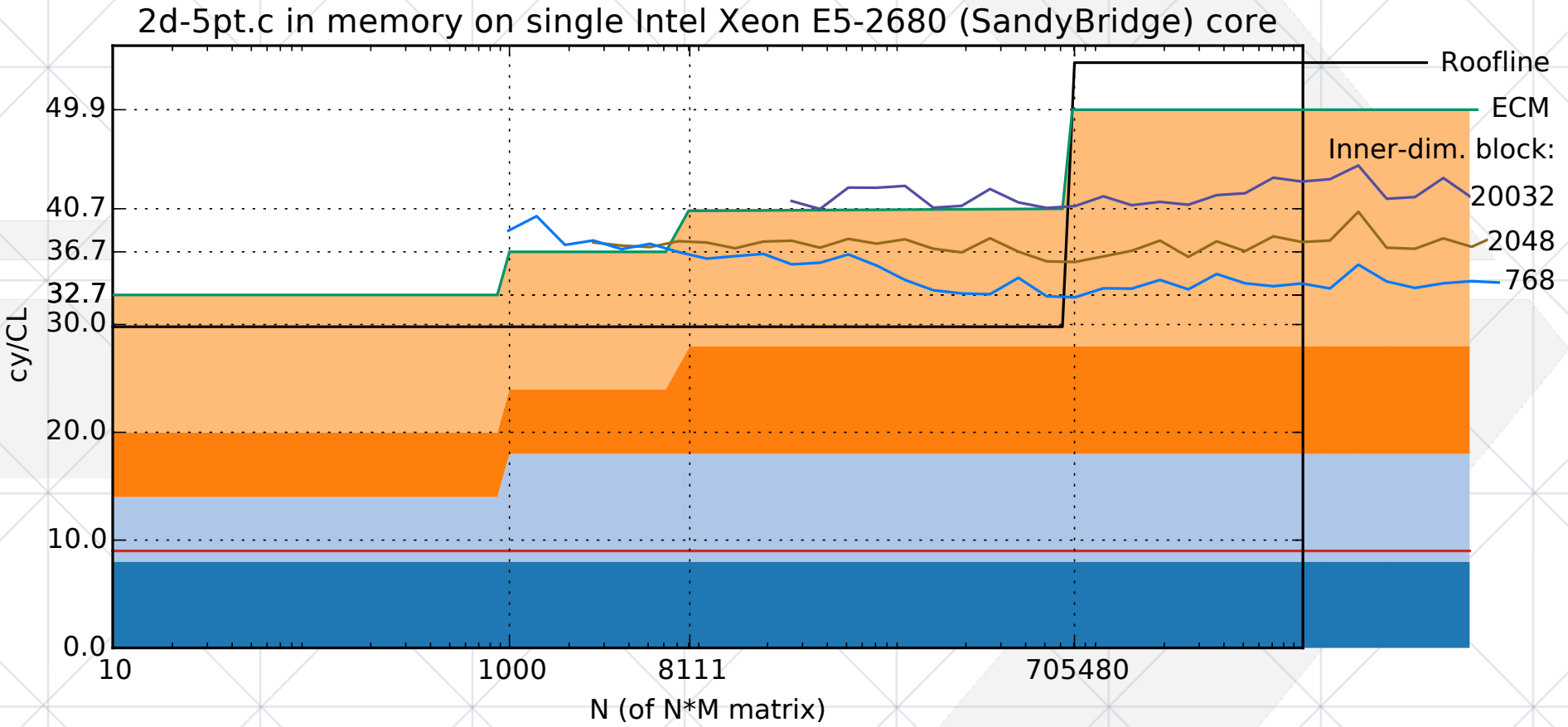
Kerncraft – Results



Kerncraft – Results



Kerncraft – Spatial Blocking



Kerncraft – Verbose Output

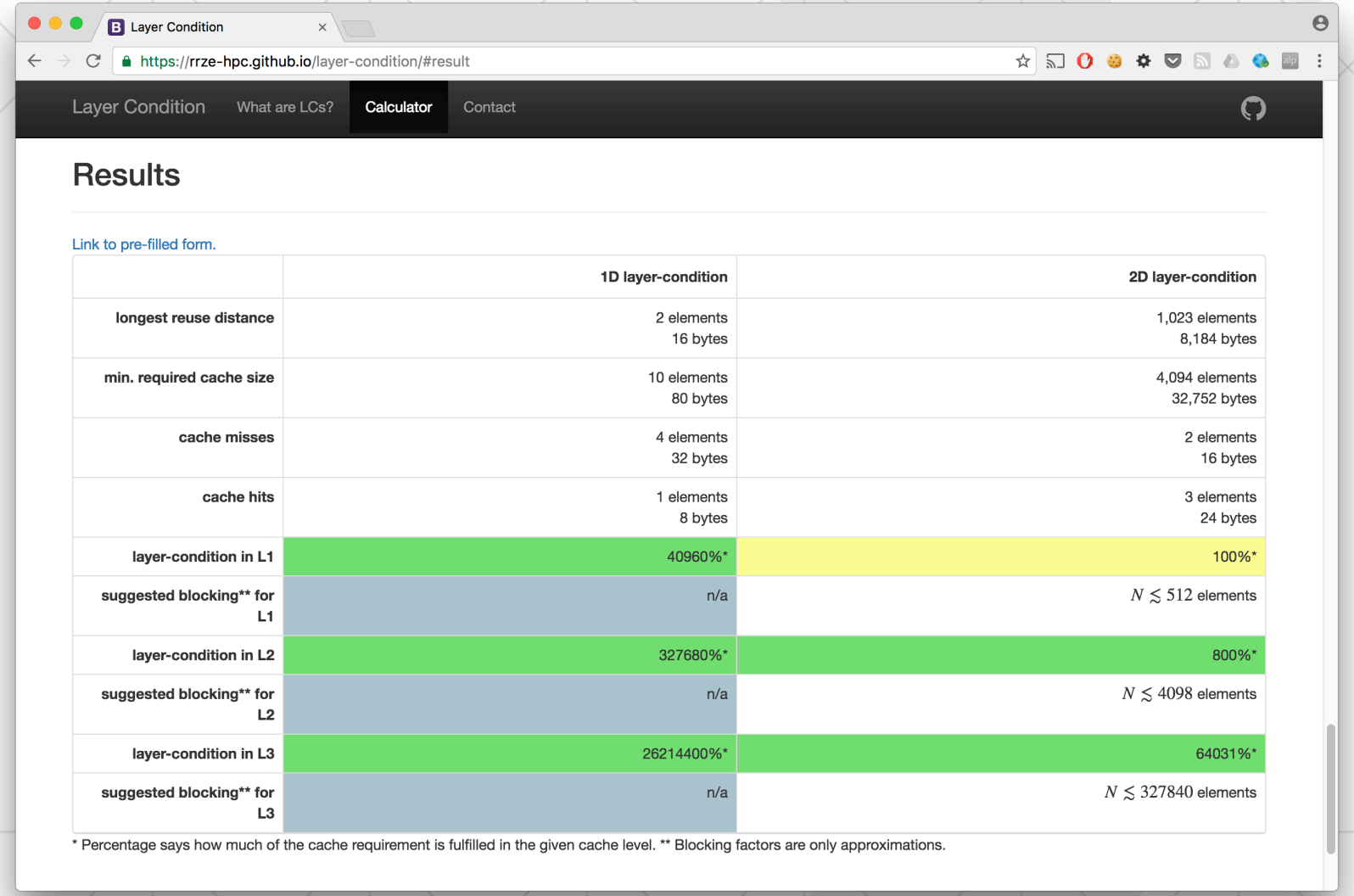
Layer condition analysis:

```
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel LC
[…]
1D Layer-Condition:
L1: unconditionally fulfilled
L2: unconditionally fulfilled
L3: unconditionally fulfilled
2D Layer-Condition:
L1: N <= 1024
L2: N <= 8192
L3: N <= 655360
$
```

Also available as web-based calculator:

https://rrze-hpc.github.io/layer-condition/#calculator

Layer Condition Calculator



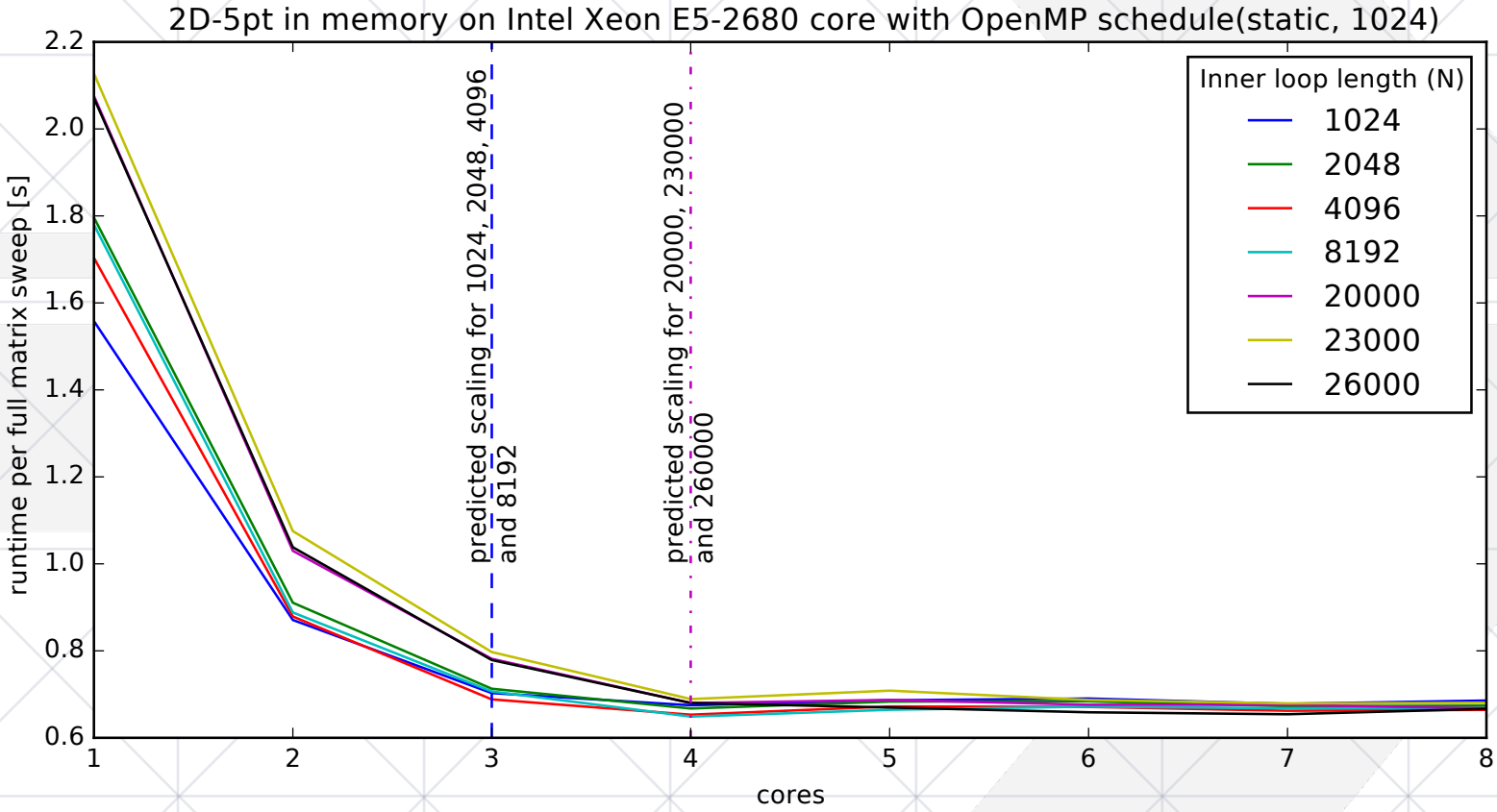
Results

Link to pre-filled form.

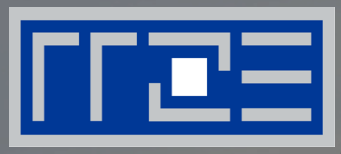
	1D layer-condition	2D layer-condition
longest reuse distance	2 elements 16 bytes	1,023 elements 8,184 bytes
min. required cache size	10 elements 80 bytes	4,094 elements 32,752 bytes
cache misses	4 elements 32 bytes	2 elements 16 bytes
cache hits	1 elements 8 bytes	3 elements 24 bytes
layer-condition in L1	40960%*	100%*
suggested blocking for L1**	n/a	$N \lesssim 512$ elements
layer-condition in L2	327680%*	800%*
suggested blocking for L2**	n/a	$N \lesssim 4098$ elements
layer-condition in L3	26214400%*	64031%*
suggested blocking for L3**	n/a	$N \lesssim 327840$ elements

* Percentage says how much of the cache requirement is fulfilled in the given cache level. ** Blocking factors are only approximations.

Kerncraft – In-Socket Scaling

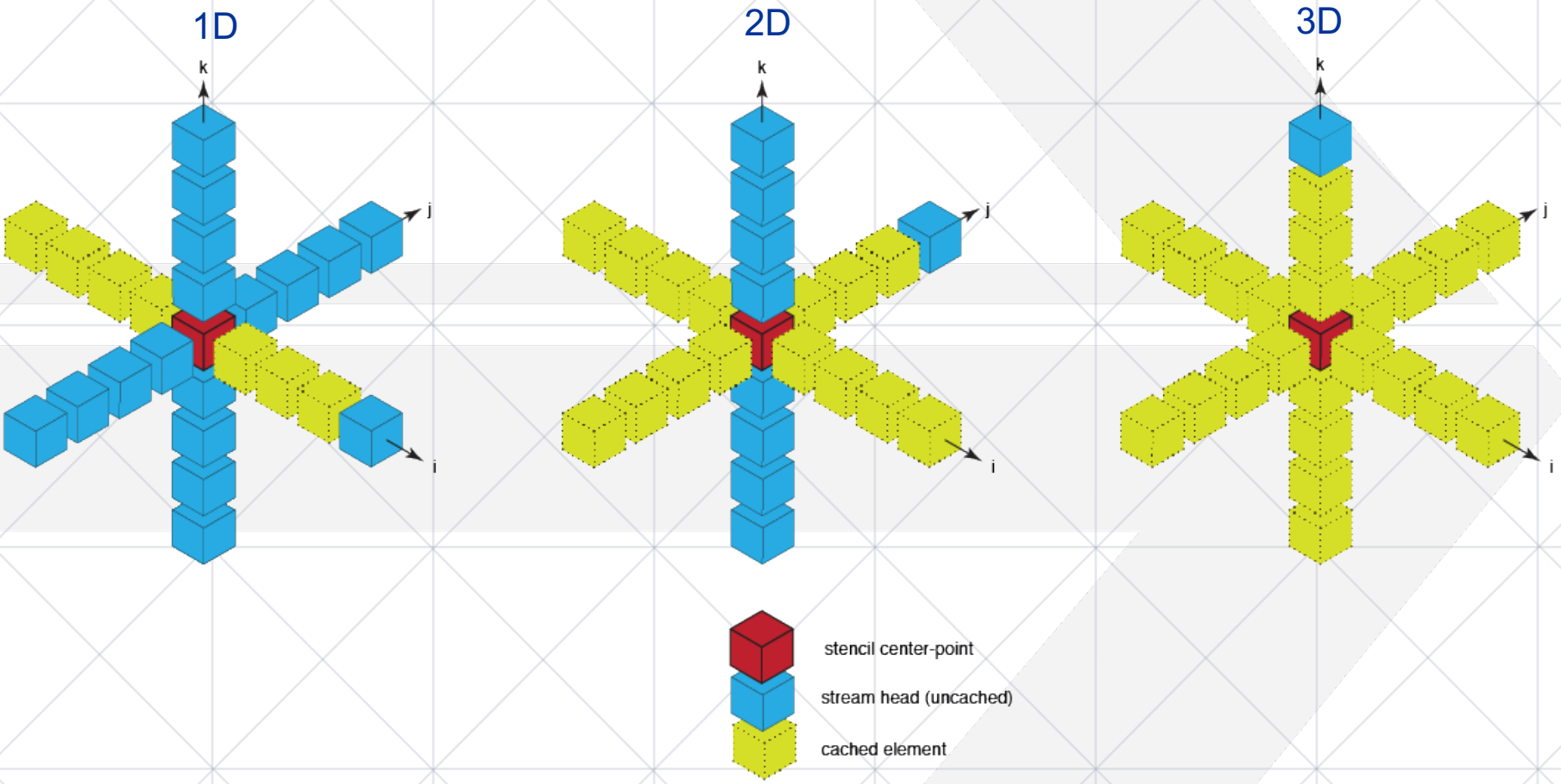


3D-LONG-RANGE EXAMPLE



```
double U[M][N][N];
double V[M][N][N];
double ROC[M][N][N];
double c0, c1, c2, c3, c4, lap;

for(int k=4; k < M-4; k++) {
    for(int j=4; j < N-4; j++) {
        for(int i=4; i < N-4; i++) {
            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 ])
                + c2 * ( V[ k ][ j ][i+2] + V[ k ][ j ][i-2])
                + c2 * ( V[ k ][j+2][ i ] + V[ k ][j-2][ i ])
                + c2 * ( V[k+2][ j ][ i ] + V[k-2][ j ][ i ])
                + c3 * ( V[ k ][ j ][i+3] + V[ k ][ j ][i-3])
                + c3 * ( V[ k ][j+3][ i ] + V[ k ][j-3][ i ])
                + c3 * ( V[k+3][ j ][ i ] + V[k-3][ 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;
}}}
```

3D-long-range Example



Kerncraft – Verbose Output

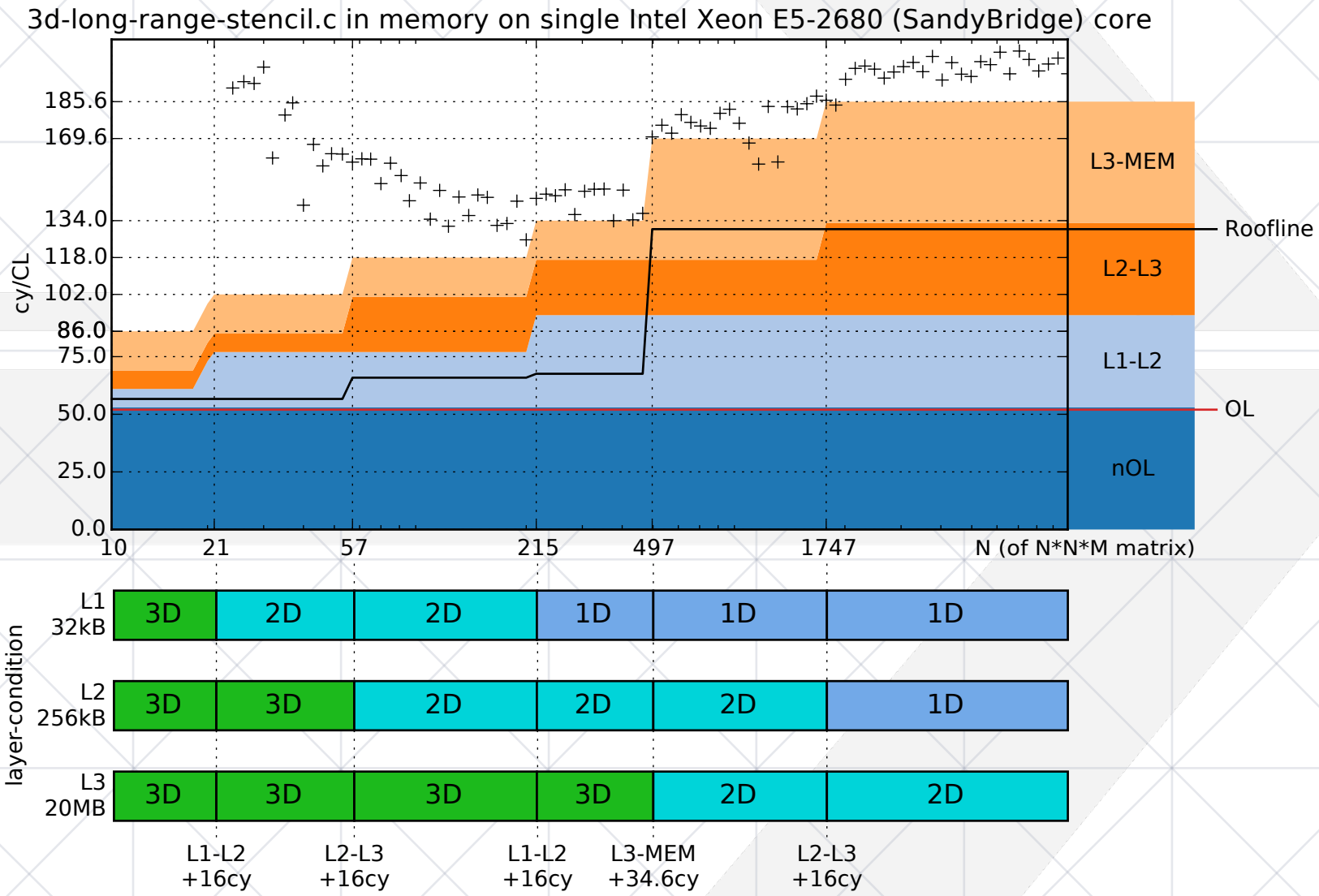
Layer condition analysis:

```
$ kerncraft --machine snb.yaml 3d-long-range.c --pmodel LC
[…]
1D Layer-Condition:
L1: unconditionally fulfilled
L2: unconditionally fulfilled
L3: unconditionally fulfilled
2D Layer-Condition:
L1: N <= 216
L2: N <= 1725
L3: N <= 137971
3D Layer-Condition:
L1: N <= 19
L2: N <= 55
L3: N <= 488
$
```

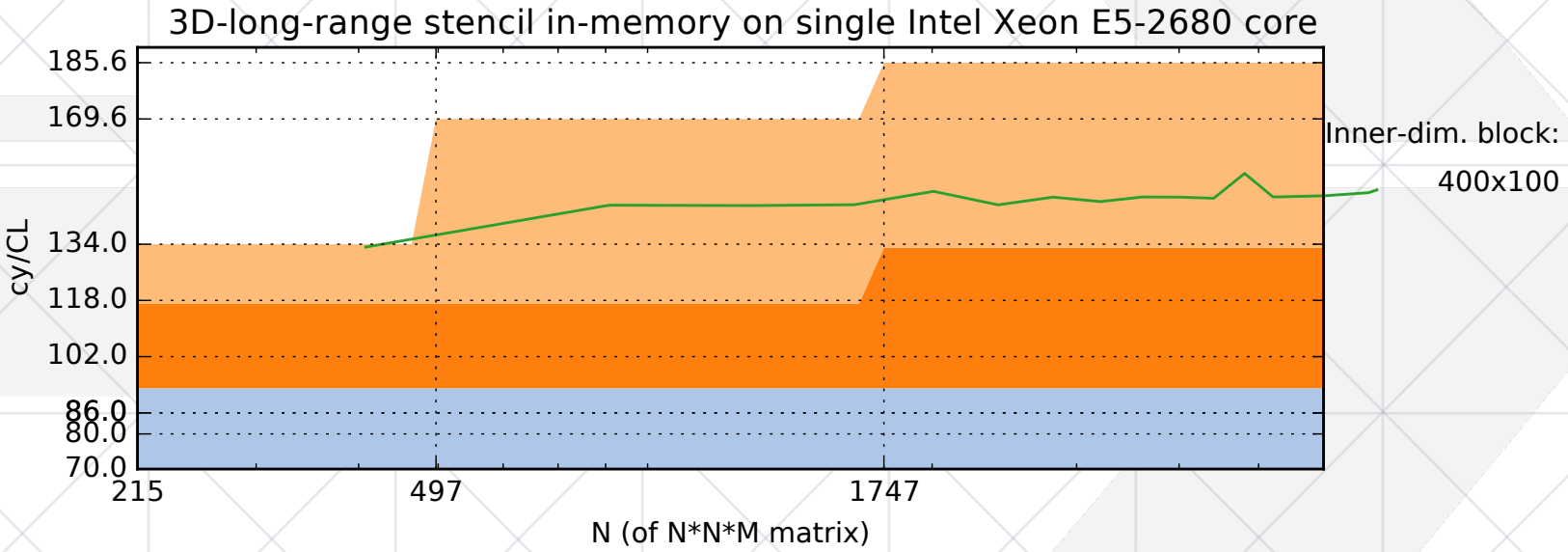
Also available as web-based calculator:

https://rrze-hpc.github.io/layer-condition/#calculator

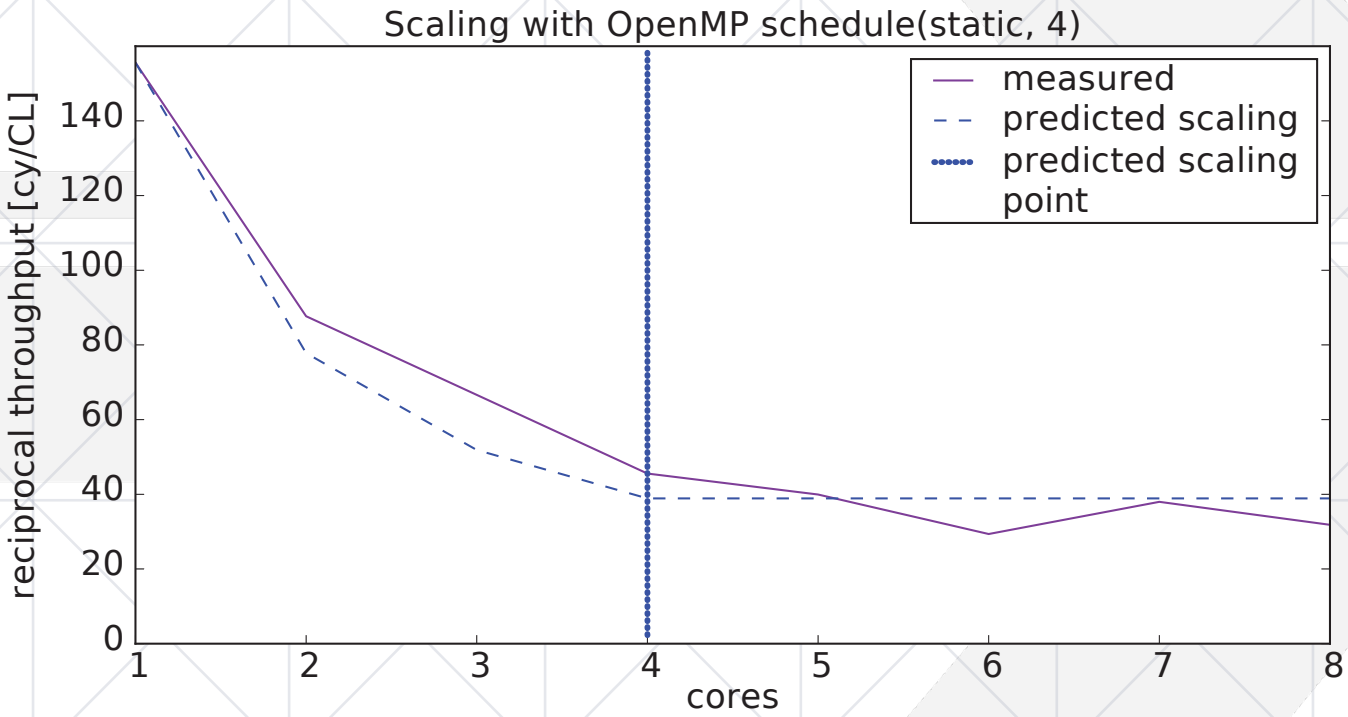
3D-long-range Example



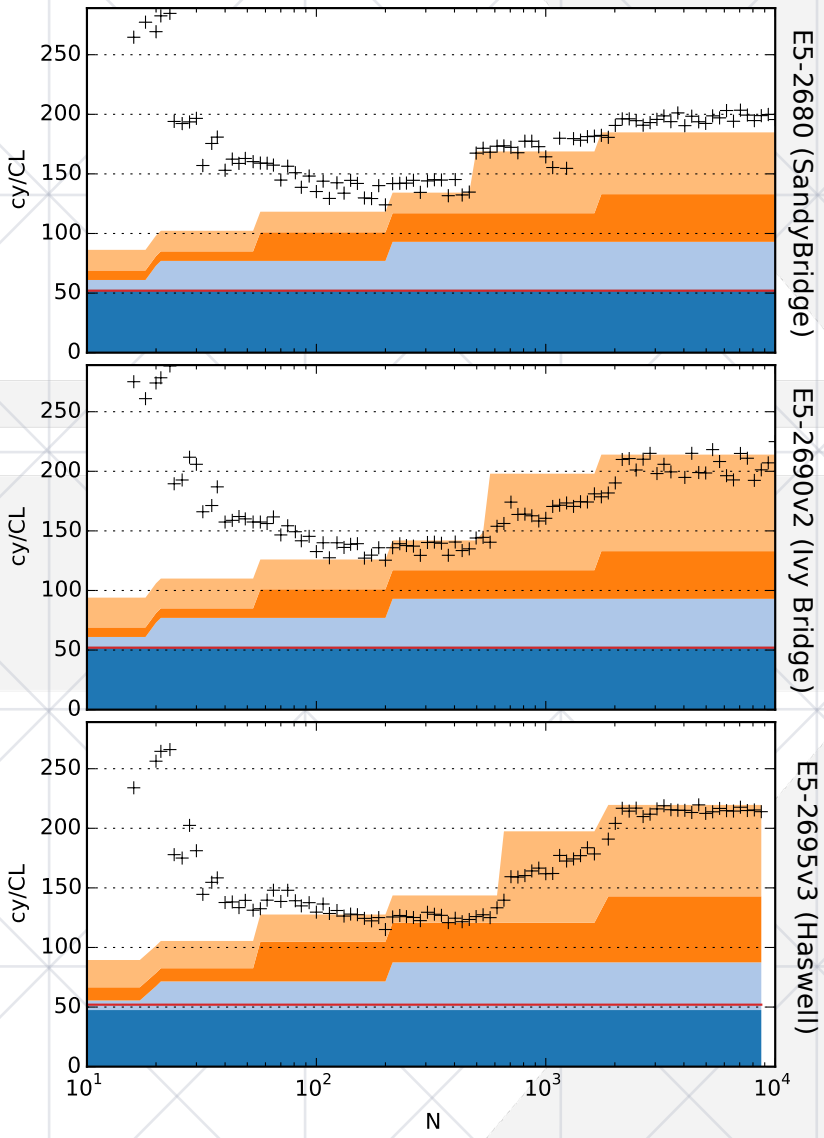
3D-long-range Example



3D-long-range Example



3D-long-range Example



Benefits

- Guiding optimizations
- Hardware-software co-design
- Energy optimized computing
- Deeper understanding of code and hardware interactions

Kerncraft...

- is a white-box utility
- takes some of the pain out of performance modeling
- is free (as in free beer and freedom)
- is NOT for inexperienced programmers
- is NOT a fully-automated jack-of-all-trades yielding better performance

Open Source


Fork me on GitHub
https://github.com/RRZE-HPC/kerncraft
Licensed under AGPLv3

Outlook

- Replacement for IACA (under investigation)
- Support for non-Intel Architectures (AMD and POWER8)
  - Depends on:
    - Support for non-inclusive cache-architectures and (work in progress)
    - ECM model support
    - Replacement for IACA
- Phenomenological performance modeling with LIKWID
- LLVM integration with polyhedral model
  - Import of kernels embedded in large code bases
  - Automatic tiling during compilation
- Irregular Performance Modeling (e.g., graph algorithms)

Thank You for Your Attention!

Julian Hammer <julian.hammer@fau.de>
RRZE High Performance Computing Group
http://www.rrze.fau.de/hpc