ERLANGEN REGIONAL COMPUTING CENTER [RRZE]

Automatic Loop Kernel Analysis and Performance Modeling with Kerncraft

Julian Hammer
Georg Hager
Jan Eitzinger
Gerhard Wellein

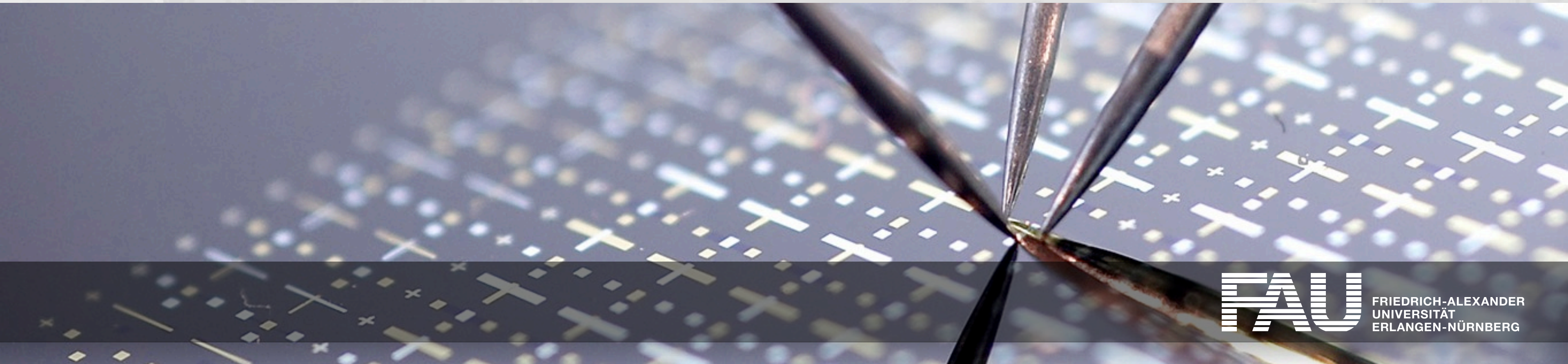
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Overview

1. Loop Kernels
2. Roofline and ECM
3. Kerncraft
    1. Overview and Structure
    2. Output and Results
4. 3D-long-range Example

LOOP KERNELS

Automatic **Loop Kernel** Analysis and Performance Modeling with Kerncraft



Loop Kernels

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

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

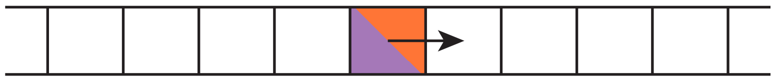
```
double a[5000][5000];
double b[5000][5000];
double s;

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;
```

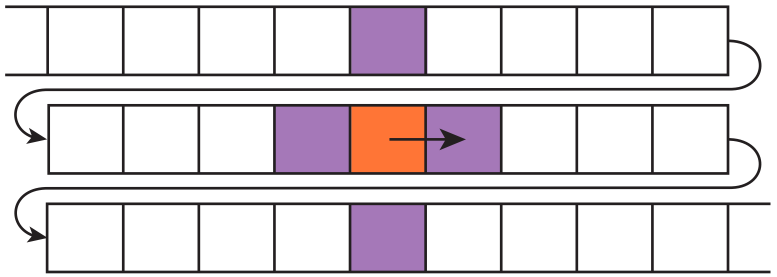
- Many inner-loop iterations
- No branching
- Access fully determined by loop counters (i.e., no irregularities)

Loop Kernels

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

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

```c
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;
```





Streaming Kernel

- Simple structure
- No data-reuse

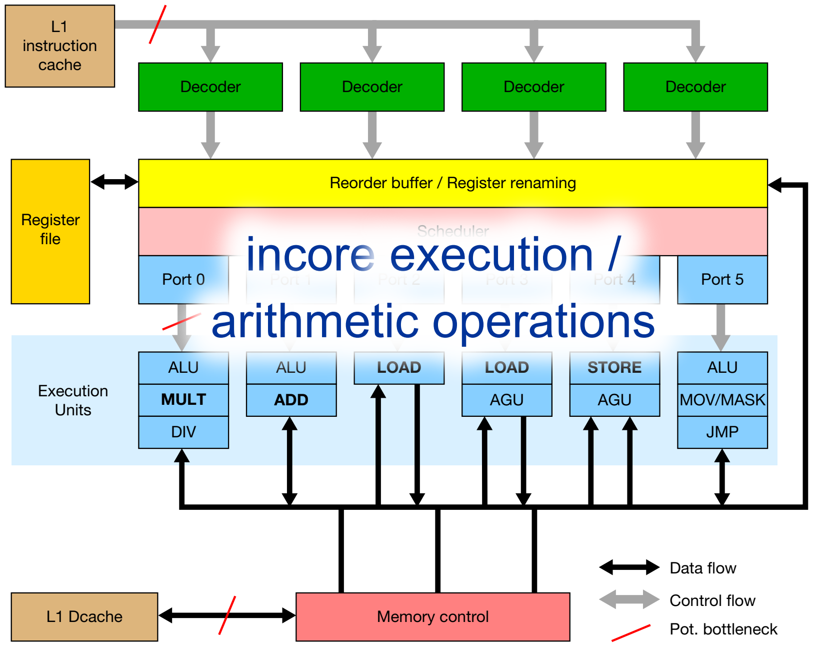
Stencil Code

- Complex Structure
- Heavy data-reuse

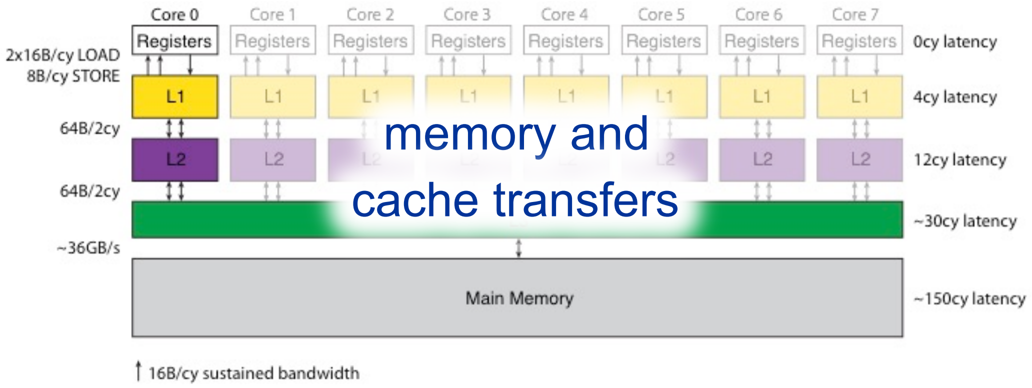
Loop Kernels

How to predict performance on complex architectures?

Two major contributions/bottlenecks:



incore execution / arithmetic operations



memory and cache transfers

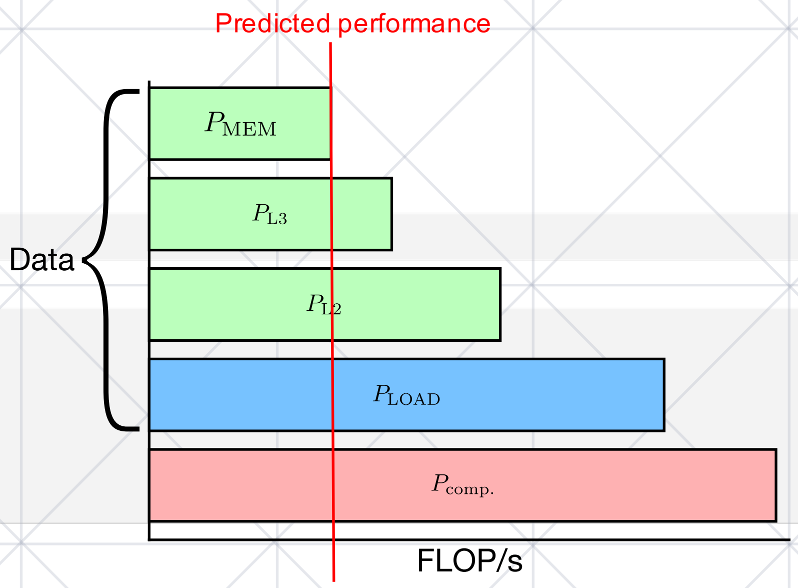
Potential Benefits

- 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

ROOFLINE AND ECM

Automatic Loop Kernel Analysis and **Performance Modeling** with Kerncraft

Roofline



Roofline

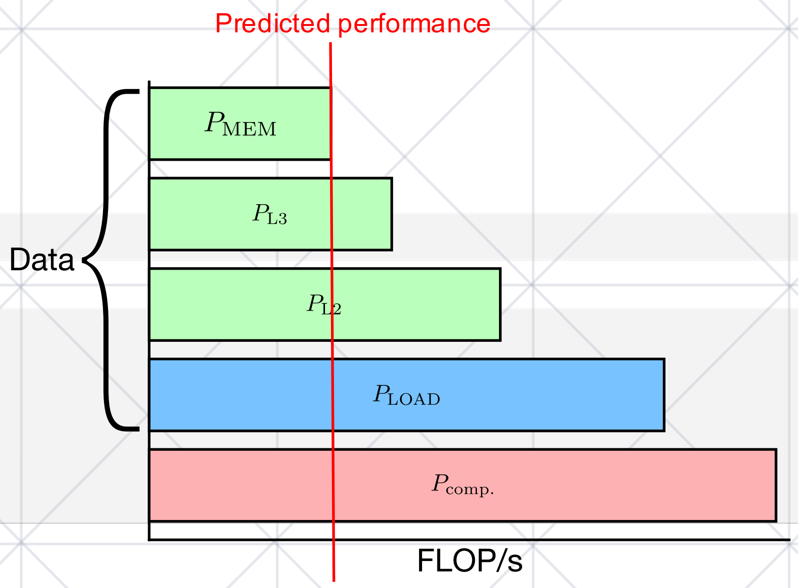
- All memory levels are separate bottlenecks

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

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

- Bandwidths are measured by suitable benchmarks

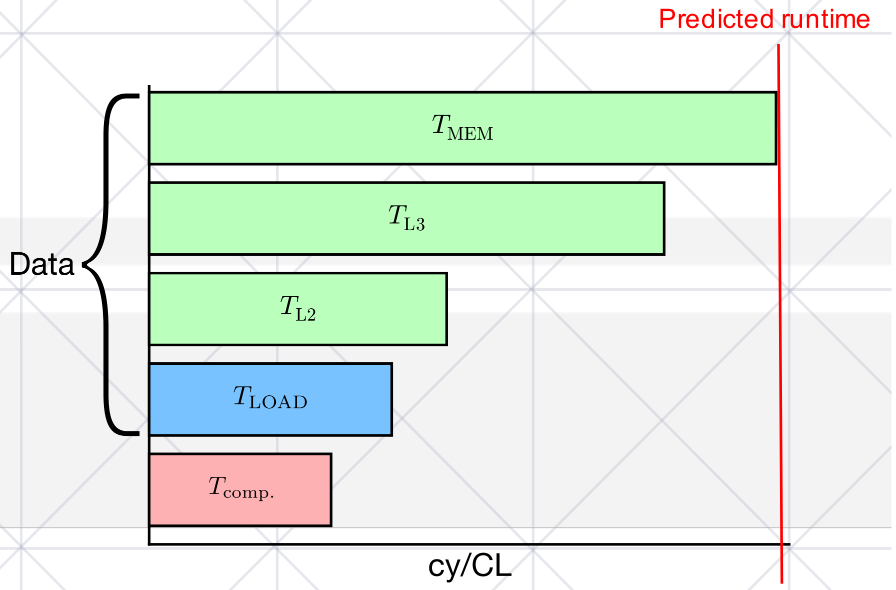
Roofline: Performance vs Time



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

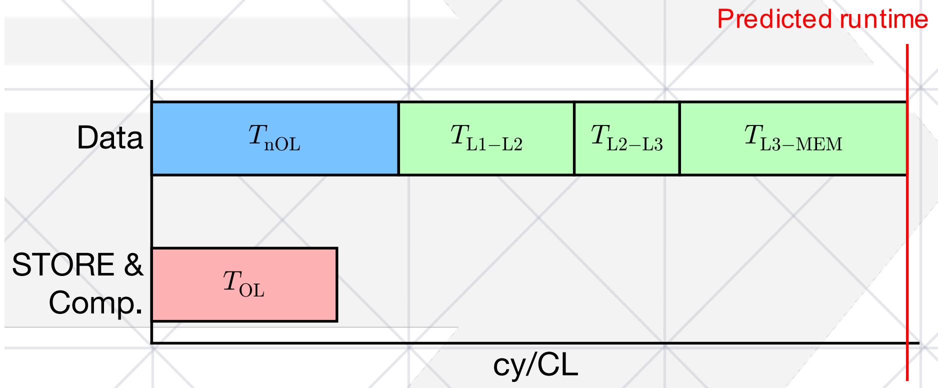
Execution-Cache-Memory (ECM) Model

- 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

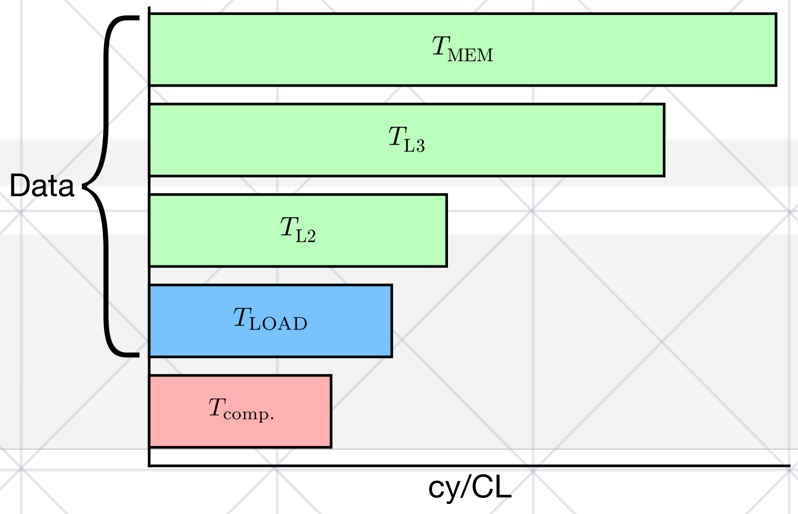
$\{ T_{OL} \,\|\, T_{nOL} \mid T_{L1\text{-}L2} \mid T_{L2\text{-}L3} \mid T_{L3\text{-}MEM} \}$

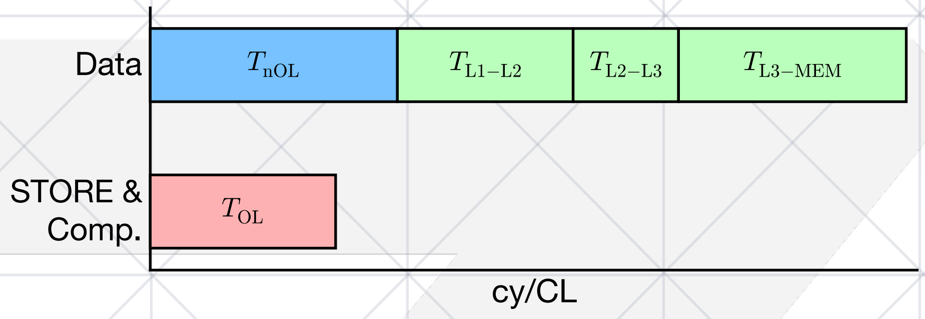
- One measured input: full-socket mem. bandwidth



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];
```

```
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;
```

STREAM Scale

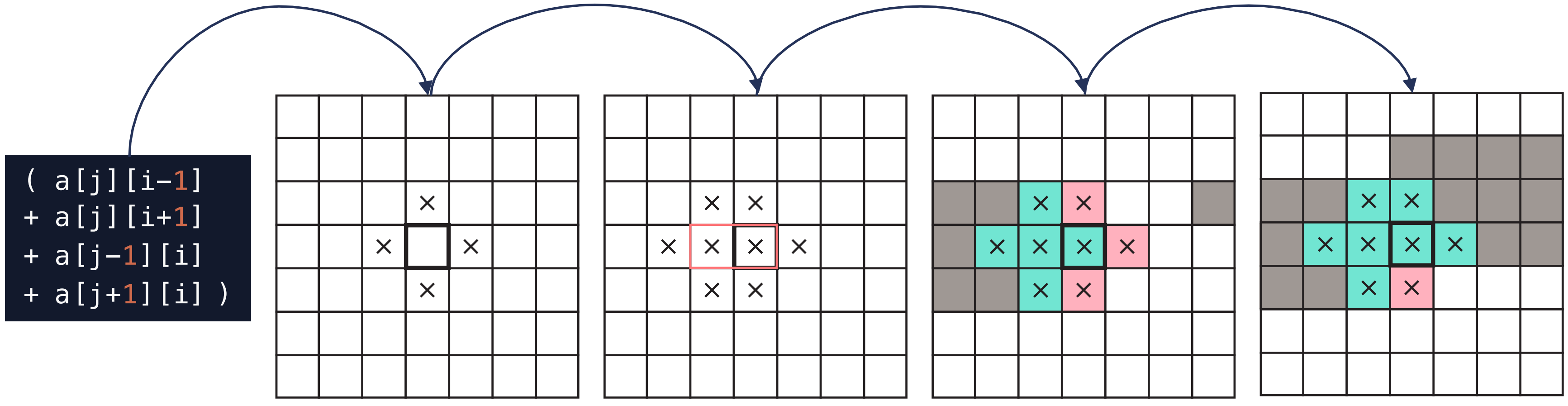
- For each cache line (8 it.):
  - 1 CL is stored
  - 8 FLOP
  - 1 CL are loaded

2D 5-point Stencil

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

Why 1-3?

Layer Conditions



```
( a[j][i-1]
+ a[j][i+1]
+ a[j-1][i]
+ a[j+1][i] )
```

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

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

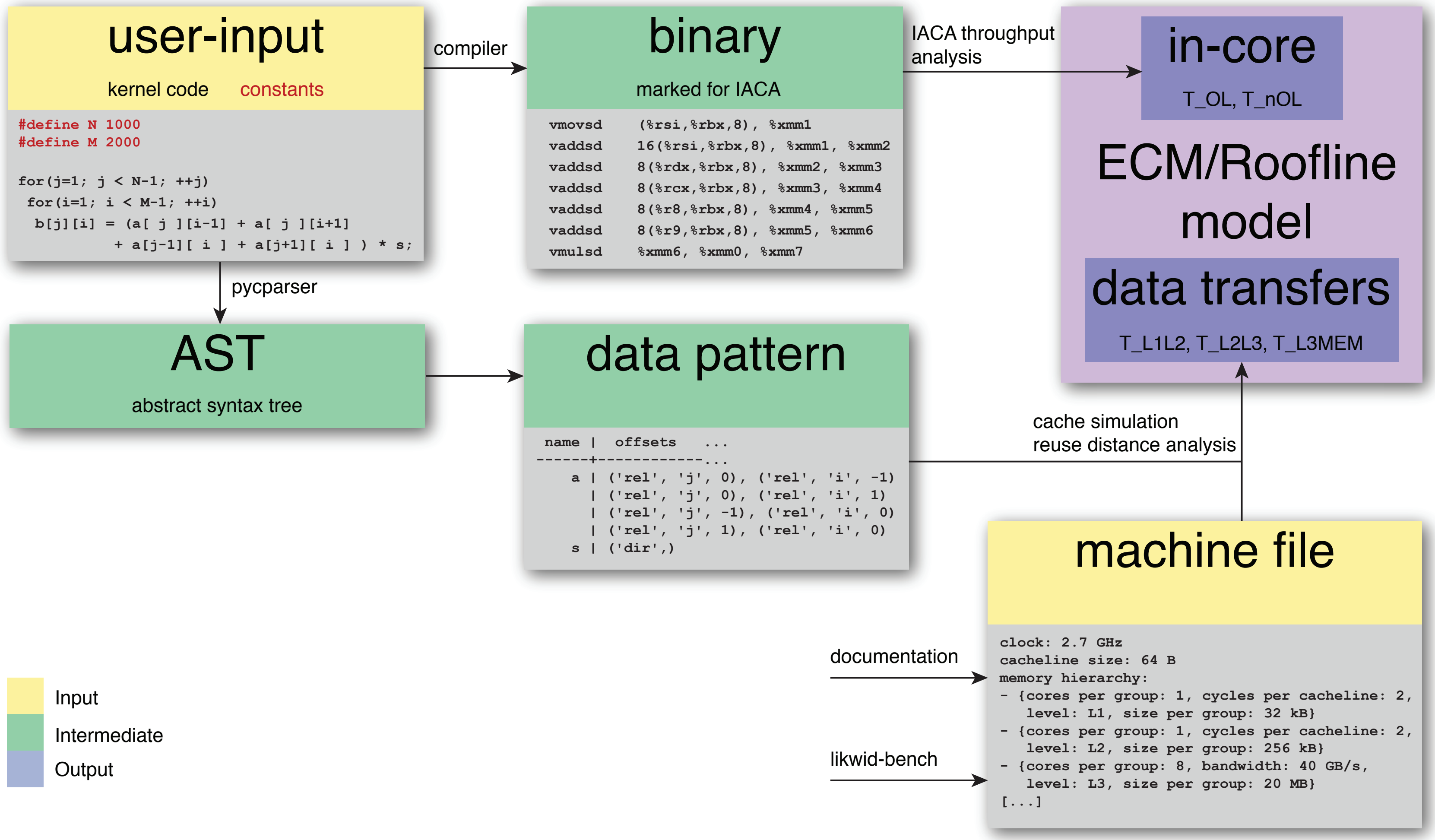
Performance Modeling

```
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

Automatic Loop Kernel Analysis and Performance Modeling **with Kerncraft**

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;
```

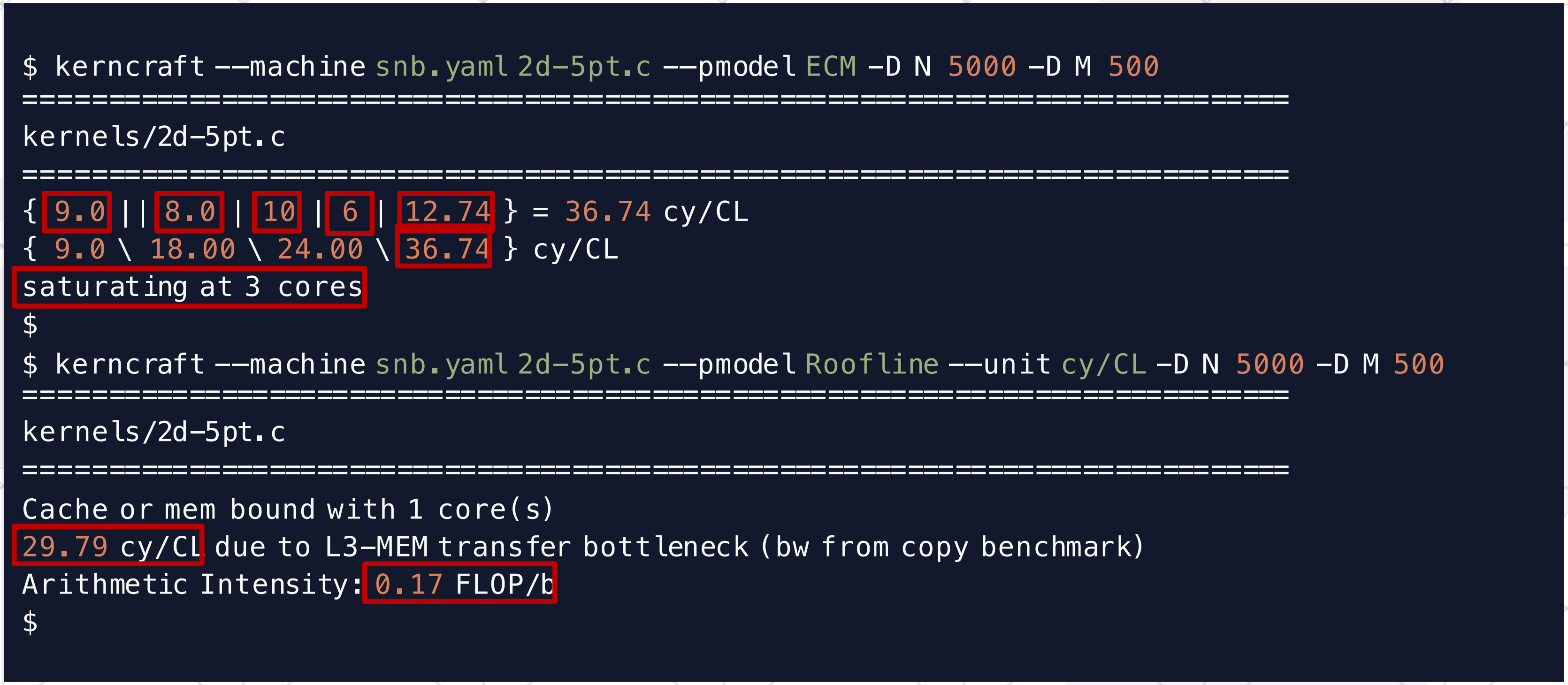
```
vmovsd    (%rsi,%rbx,8), %xmm1
vaddsd    16(%rsi,%rbx,8), %xmm1, %xmm2
vaddsd    8(%rdx,%rbx,8), %xmm2, %xmm3
vaddsd    8(%rcx,%rbx,8), %xmm3, %xmm4
vaddsd    8(%r8,%rbx,8), %xmm4, %xmm5
vaddsd    8(%r9,%rbx,8), %xmm5, %xmm6
vmulsd    %xmm6, %xmm0, %xmm7
```

```
 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',)
```

```
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}
[...]
```

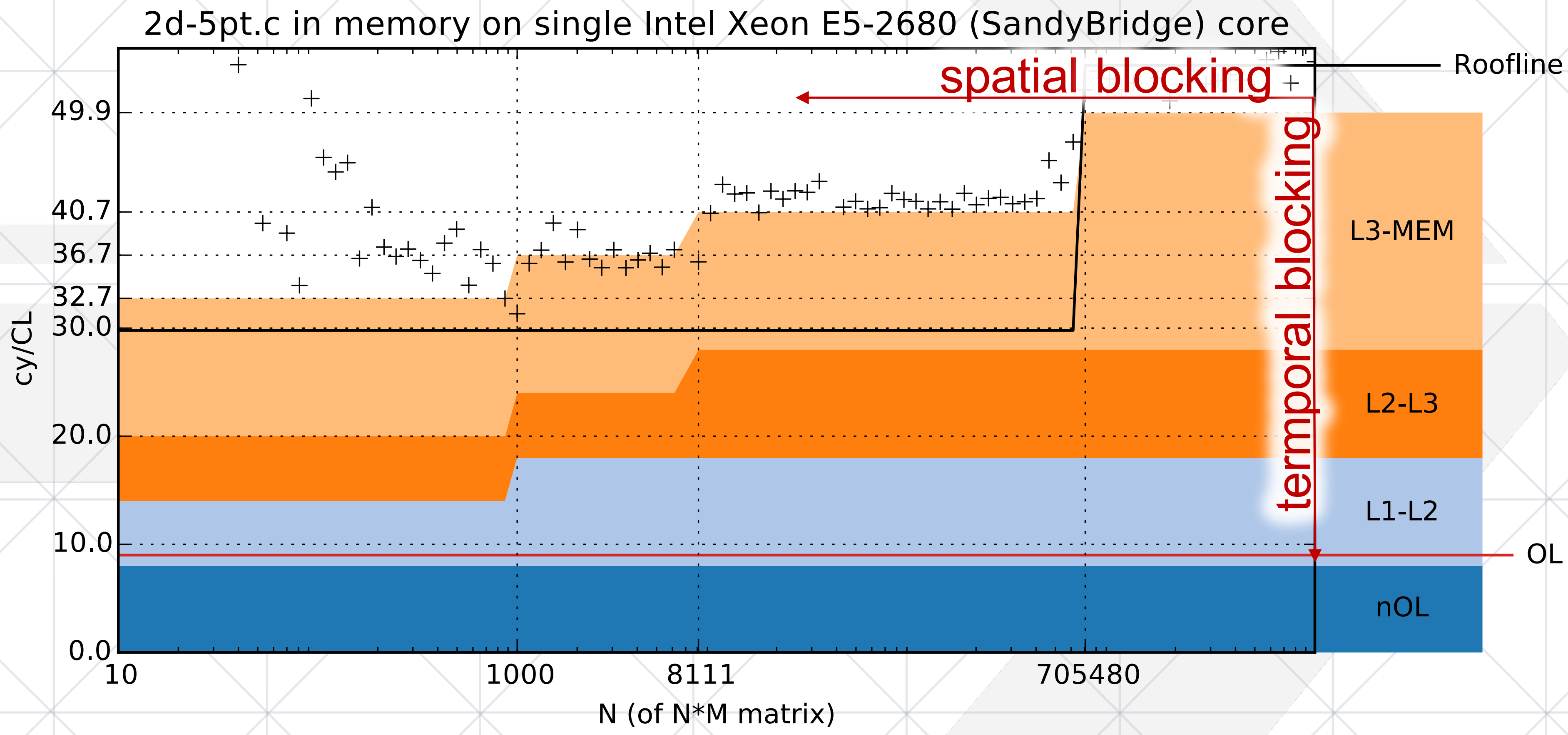
Kerncraft – Output

ECM model: $\{ T_{OL} \parallel T_{nOL} \mid T_{L1\text{-}L2} \mid T_{L2\text{-}L3} \mid T_{L3\text{-}MEM} \}$



```
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel ECM -D N 5000 -D M 500
==============================================================================
kernels/2d-5pt.c
==============================================================================
{ 9.0 || 8.0 | 10 | 6 | 12.74 } = 36.74 cy/CL
{ 9.0 \ 18.00 \ 24.00 \ 36.74 } cy/CL
saturating at 3 cores
$
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel Roofline --unit cy/CL -D N 5000 -D M 500
==============================================================================
kernels/2d-5pt.c
==============================================================================
Cache or mem bound with 1 core(s)
29.79 cy/CL due to L3-MEM transfer bottleneck (bw from copy benchmark)
Arithmetic Intensity: 0.17 FLOP/b
$
```

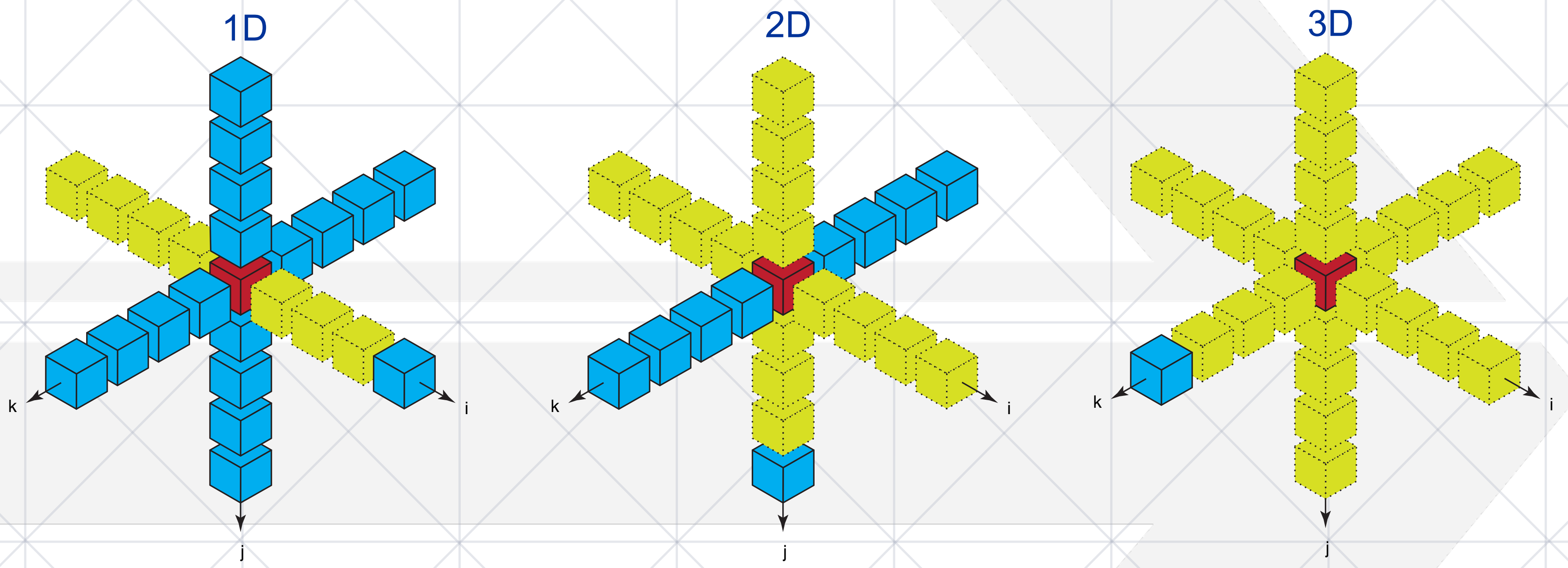
Kerncraft – Results

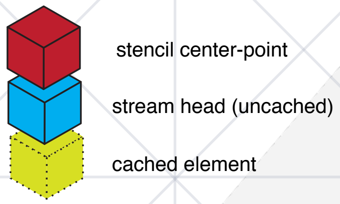


3D-LONG-RANGE EXAMPLE

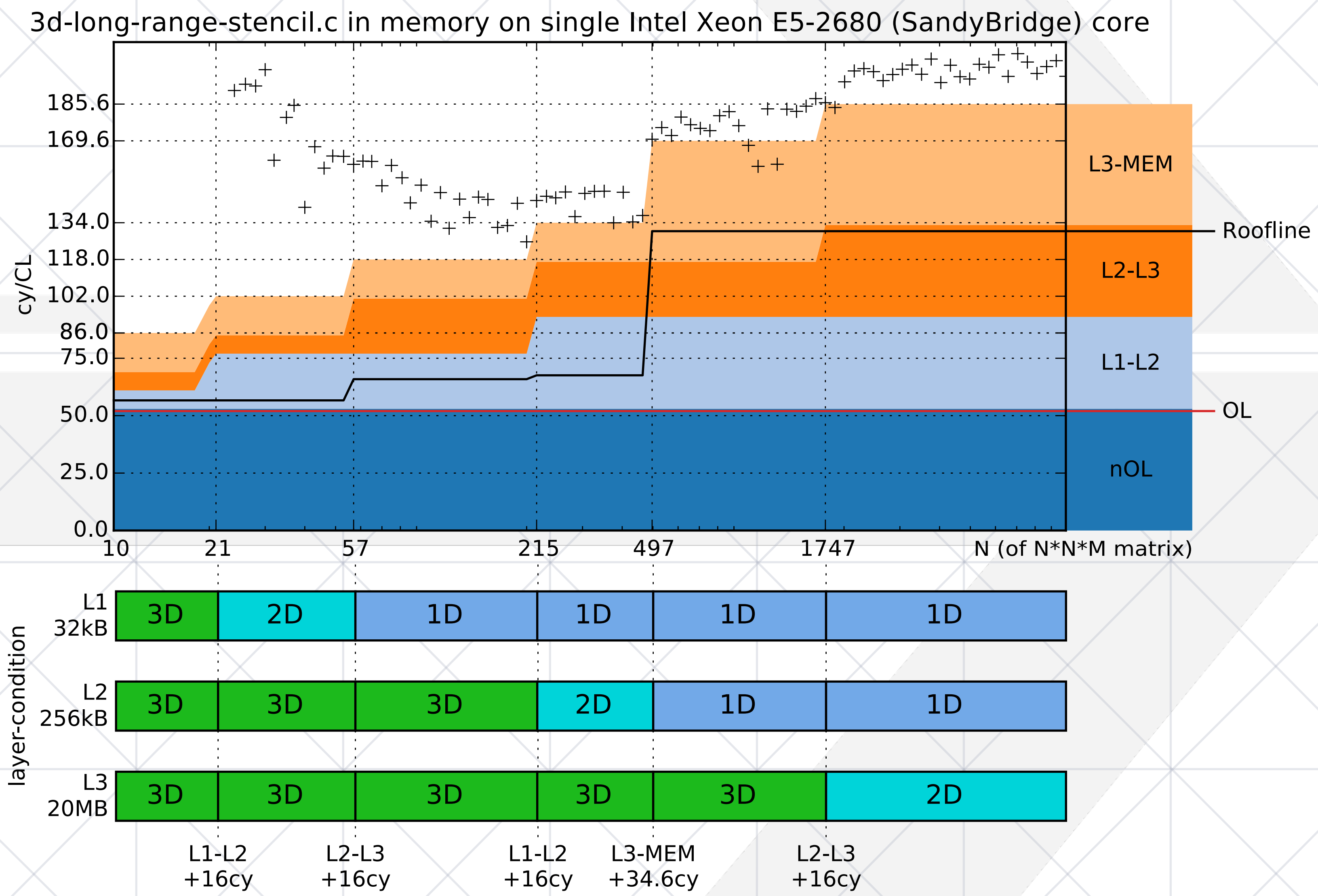
```
double U[M][N][N];
double V[M][N][N];
double ROC[M][N][N];
double c0, c1, c2, c3, c4, lap;

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

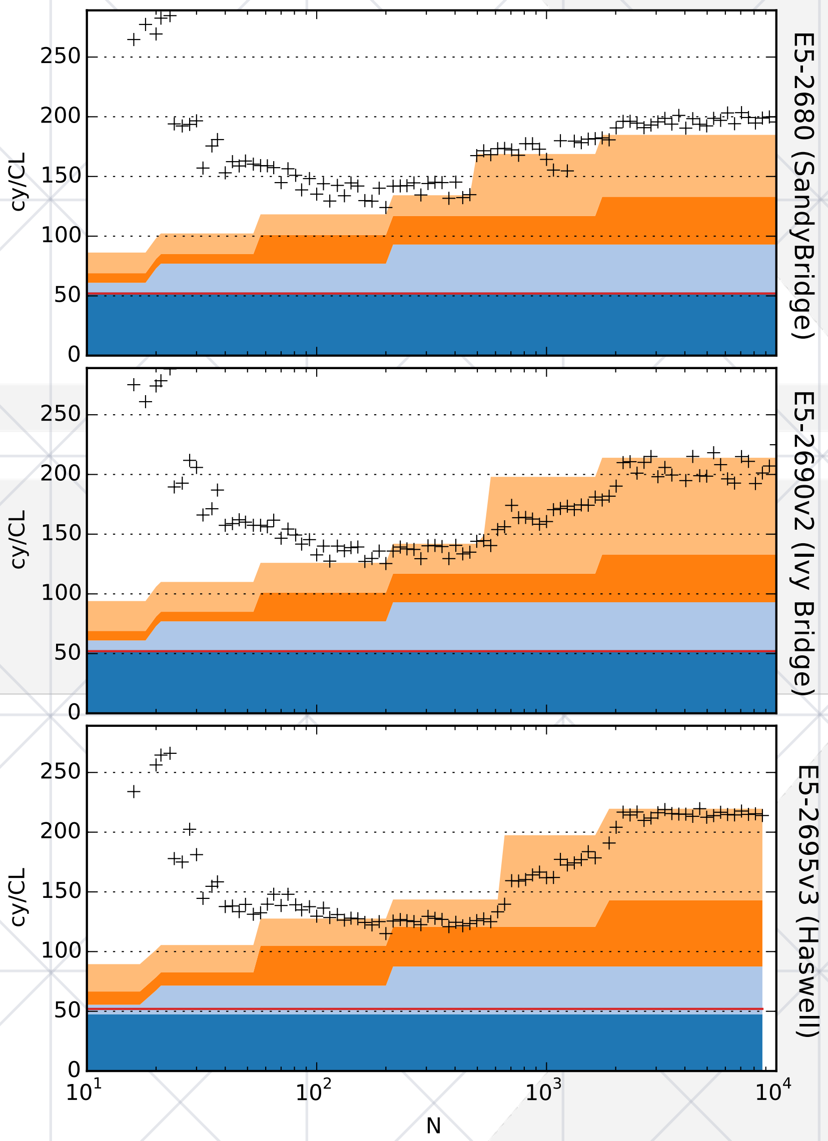




3D-long-range Example



3D-long-range Example



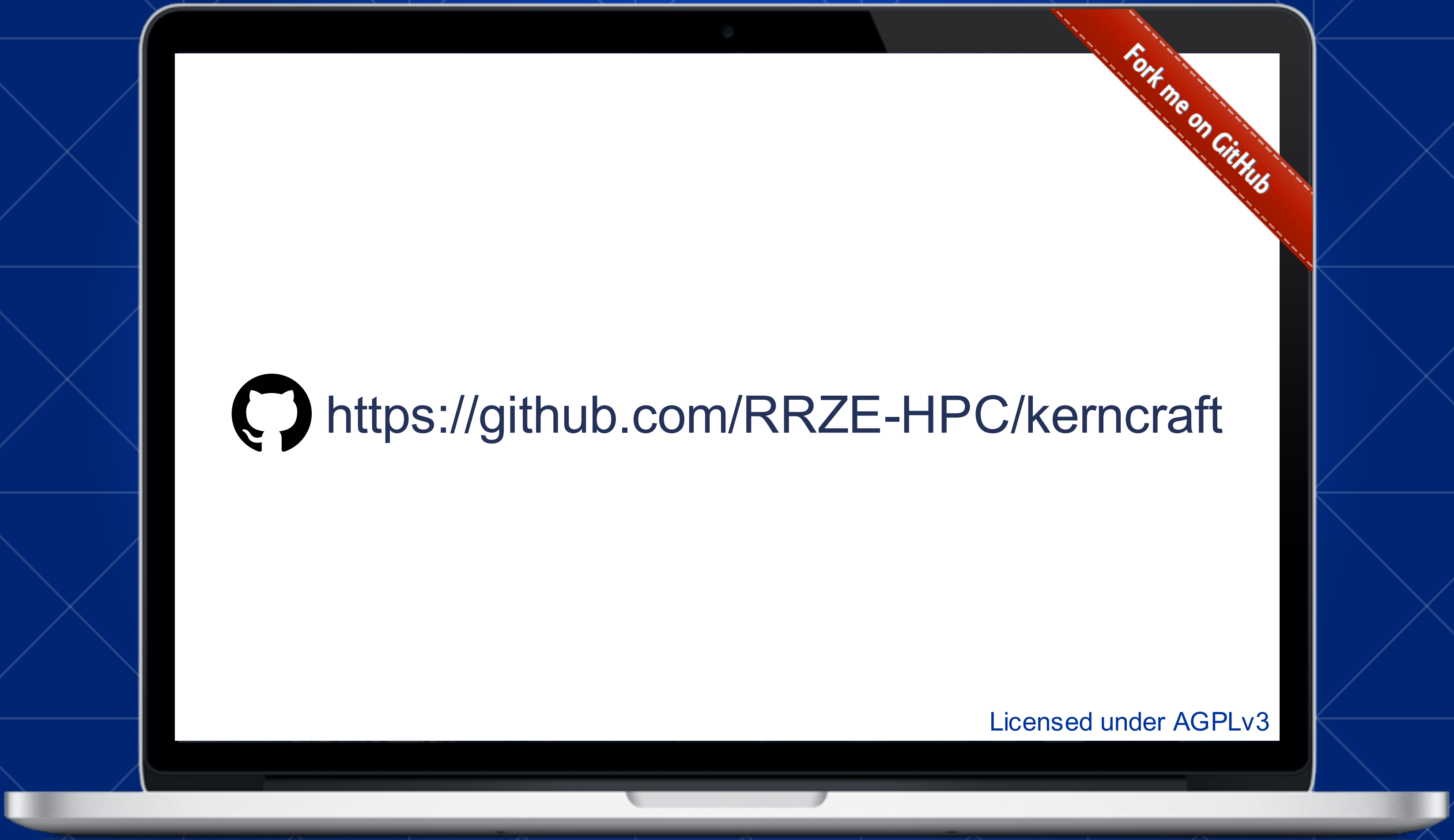
Benefits

- 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



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

Licensed under AGPLv3

Further Work

- Support for other compilers than ICC (done)
- Lift some code restrictions (partially done)
- Replacement for IACA (under investigation)
- Support for non-Intel Architectures

ERLANGEN REGIONAL
COMPUTING CENTER [RRZE]

Thank You for Your Attention!

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

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG