Basics of performance modeling for numerical applications: Roofline model and beyond

Georg Hager, Jan Treibig, Gerhard Wellein

SPPEXA PhD Seminar
RRZE
April 30, 2014

Prelude:
Scalability 4 the win!

Scalability Myth: Code scalability is the key issue

Lore 1

In a world of highly parallel computer architectures only highly scalable codes will survive

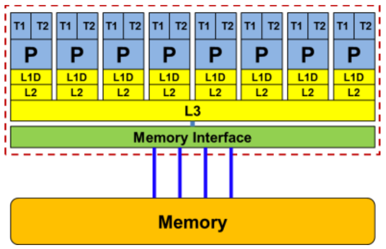
Lore 2

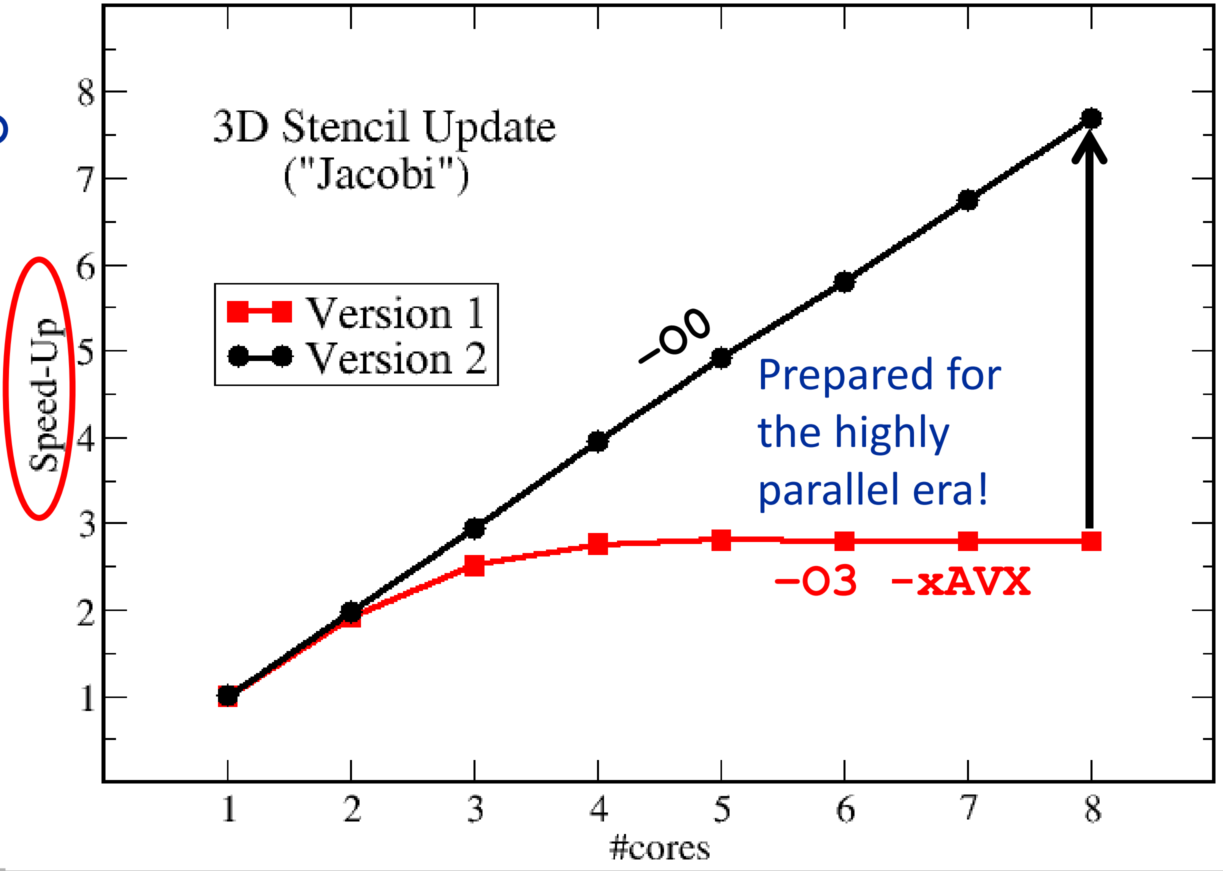
Single core performance no longer matters since we have so many of them and use scalable codes

Scalability Myth: Code scalability is the key issue

```
!$OMP PARALLEL DO
do k = 1 , Nk
  do j = 1 , Nj; do i = 1 , Ni
     y(i,j,k)= b*(  x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                    x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
    enddo; enddo
enddo
!$OMP END PARALLEL DO
```

Changing only a the compile options makes this code scalable on an 8-core chip

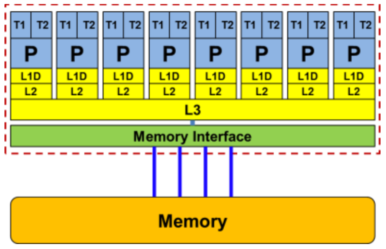


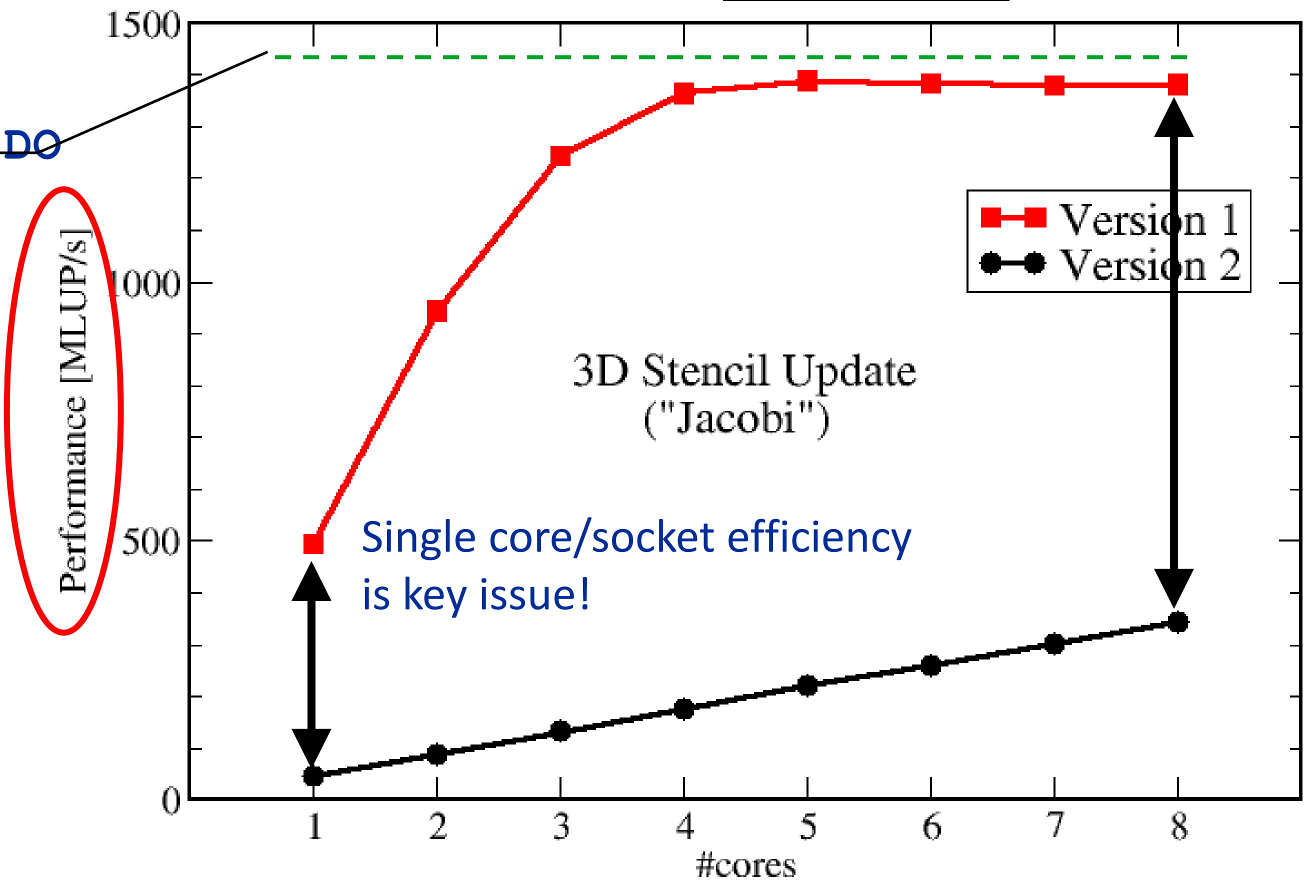


Scalability Myth: Code scalability is the key issue

```
!$OMP PARALLEL DO
do k = 1 , Nk
  do j = 1 , Nj; do i = 1 , Ni
     y(i,j,k)= b*(   x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                     x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
    enddo; enddo
enddo
                 L DO
```

Upper limit from simple performance model:
35 GB/s & 24 Byte/update





Questions to ask in high performance computing

- **Do I understand the performance behavior of my code?**
  - Does the performance match a model I have made?

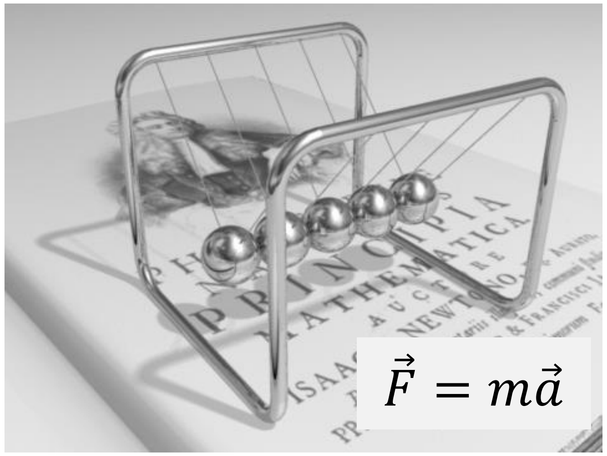
- **What is the optimal performance for my code on a given machine?**
  - High Performance Computing == Computing at the bottleneck

- **Can I change my code so that the "optimal performance" gets higher?**
  - Circumventing/ameliorating the impact of the bottleneck

- **My model does not work – what's wrong?**
  - This is the good case, because you learn something
  - Performance monitoring / microbenchmarking may help clear up the situation

An example from physics

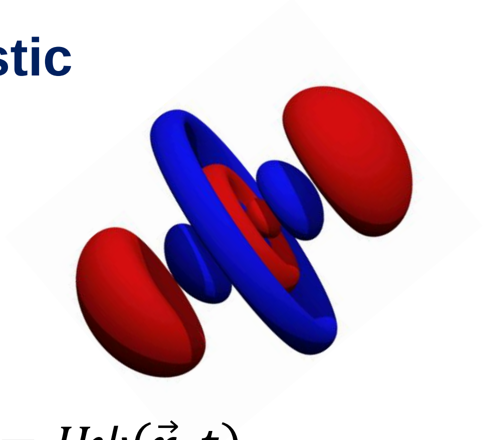
Newtonian mechanics



$$\vec{F} = m\vec{a}$$

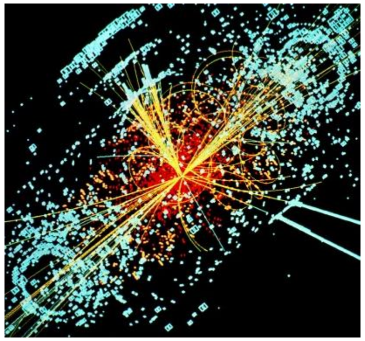
Fails @ small scales!

Nonrelativistic quantum mechanics



$$i\hbar\frac{\partial}{\partial t}\psi(\vec{r},t) = H\psi(\vec{r},t)$$

Fails @ even smaller scales!



Relativistic quantum field theory

$$U(1)_Y \otimes SU(2)_L \otimes SU(3)_c$$

Consequences

- If models fail, we learn more
- A simple model can get us very far before we need to refine

A little bit of modern computer architecture

Core

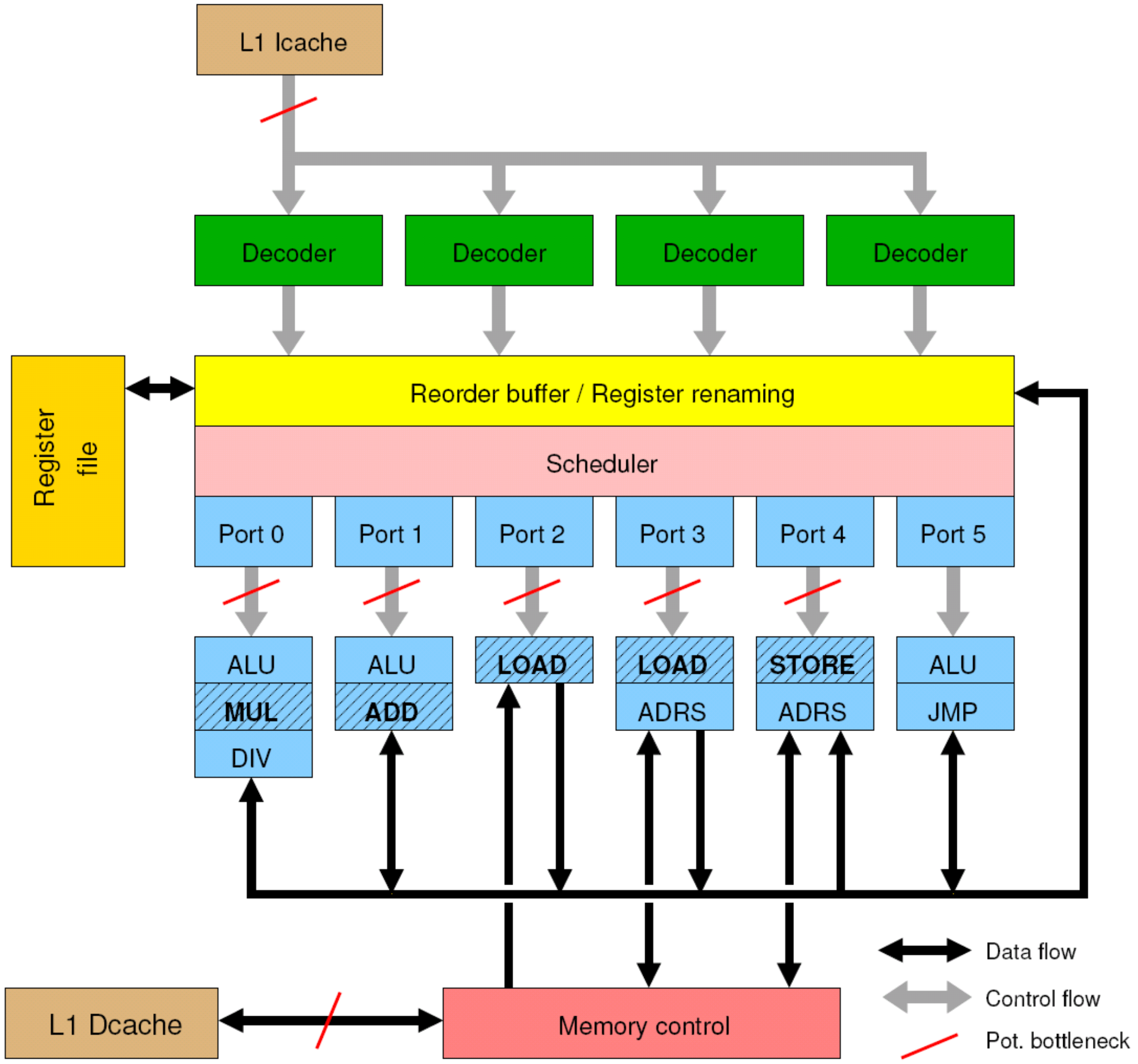
Data transfer

Topology

Bottlenecks

A typical modern processor core

- **Similar design on all modern systems**

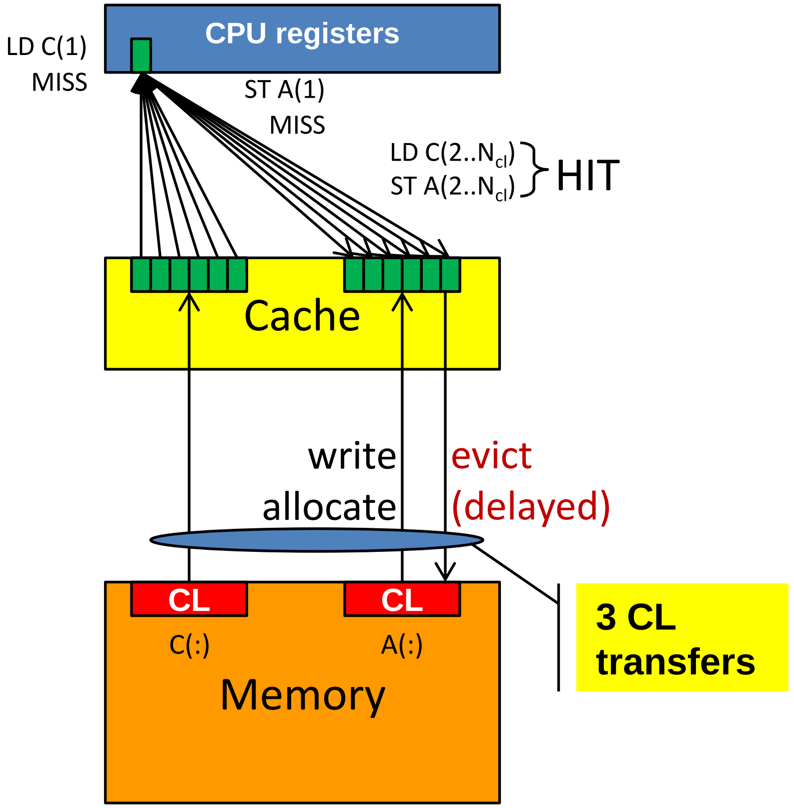


Registers and caches: Data transfers in a memory hierarchy

- How does data travel from memory to the CPU and back?

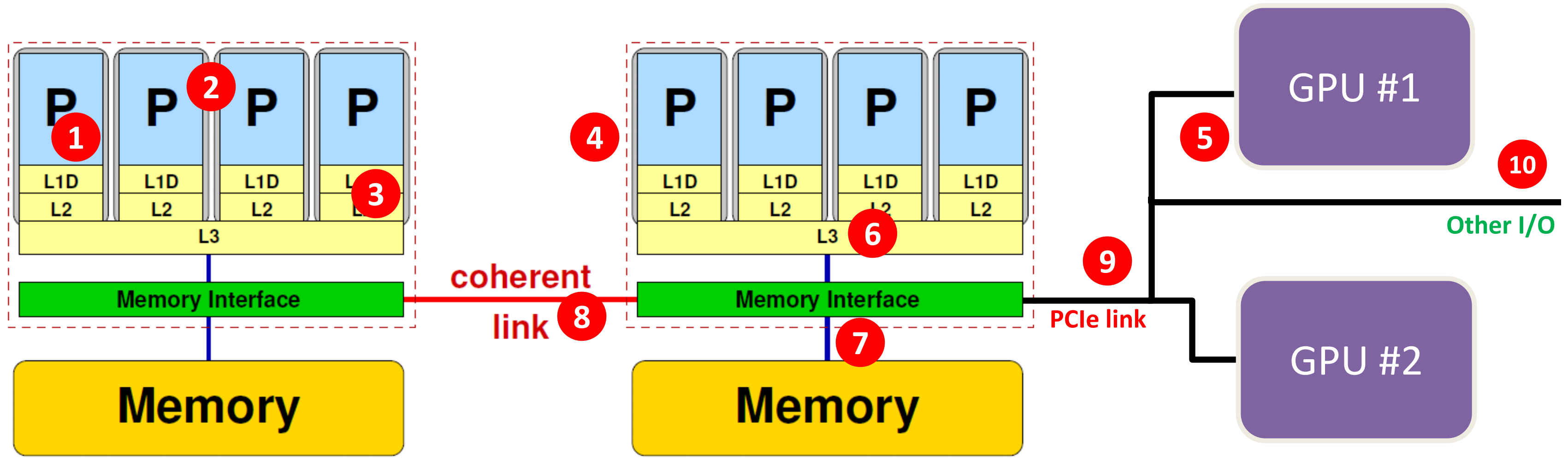
- Remember: Caches are organized in cache lines (e.g., 64 bytes)
- Only complete cache lines are transferred between memory hierarchy levels (except registers)
- MISS: Load or store instruction does not find the data in a cache level
  → CL transfer required

- Example: Array copy `A(:)=C(:)`



Parallelism in a modern compute node

- **Parallel and shared resources within a shared-memory node**



Parallel resources:

- Execution/SIMD units 1
- Cores 2
- Inner cache levels 3
- Sockets / ccNUMA domains 4
- Multiple accelerators 5

Shared resources ("bottlenecks"):

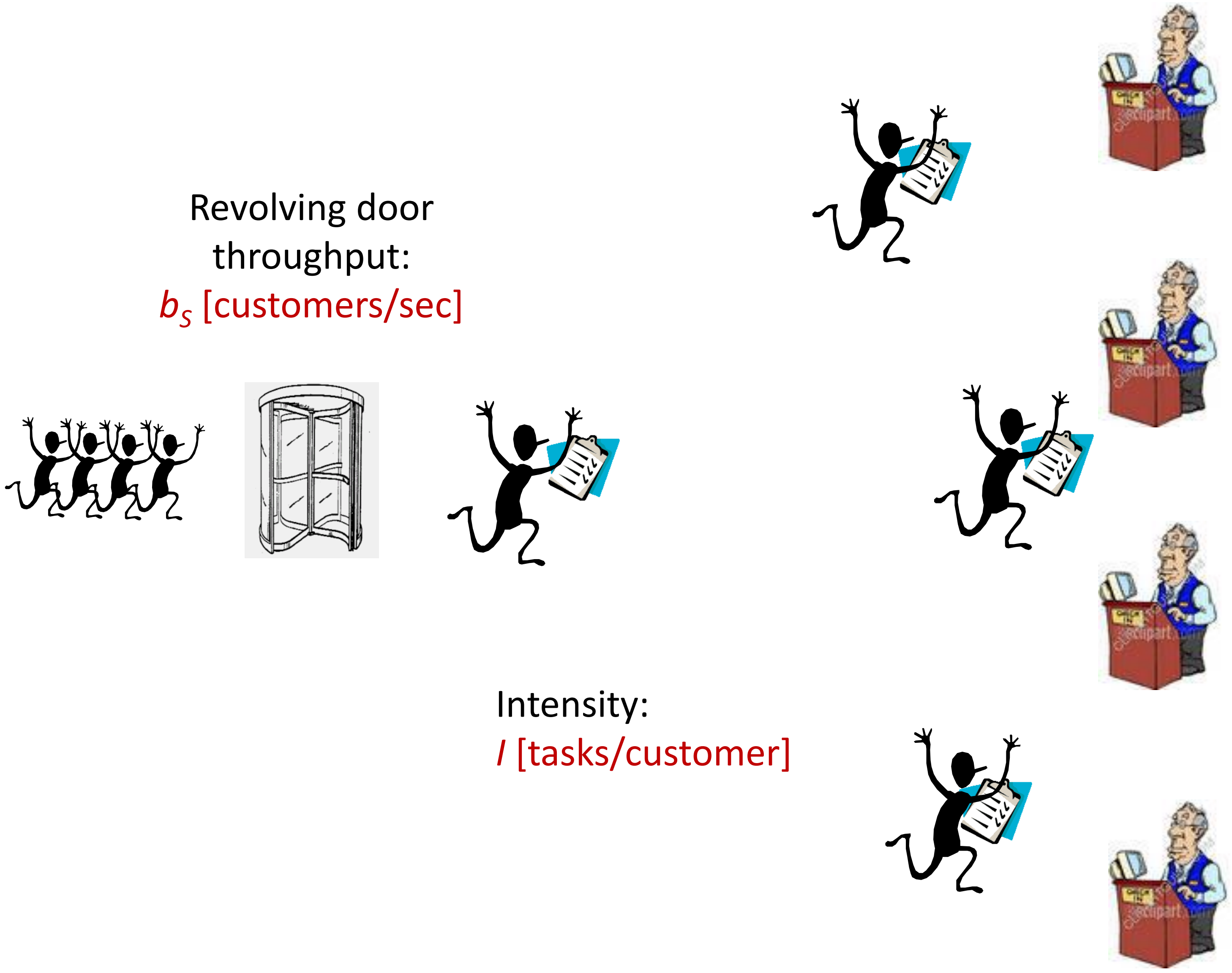
- Outer cache level per socket 6
- Memory bus per socket 7
- Intersocket link 8
- PCIe bus(es) 9
- Other I/O resources 10

Where is the bottleneck for your application?

“Simple” performance modeling: The Roofline Model

Loop-based performance modeling: Execution vs. data transfer
Example: A 3D Jacobi solver
Model-guided optimization

Prelude: Modeling customer dispatch in a bank



Revolving door throughput:
b_S [customers/sec]

Intensity:
I [tasks/customer]

Processing capability:
P_{max} [tasks/sec]

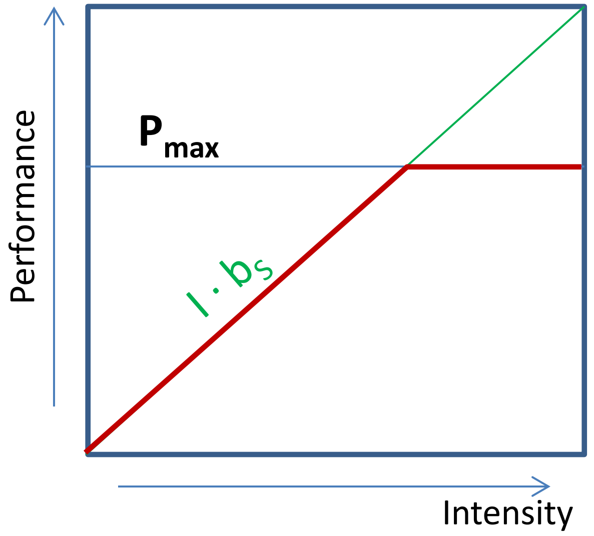
Prelude: Modeling customer dispatch in a bank

How fast can tasks be processed? *P* [tasks/sec]

- **The bottleneck is either**
  - The service desks (max. tasks/sec)
  - The revolving door (max. customers/sec)

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

- **This is the "Roofline Model"**
  - High intensity: P limited by "execution"
  - Low intensity: P limited by "bottleneck"



The Roofline Model for loop code execution[1,2]

1. **P_{max} = Applicable peak performance of a loop, assuming that data comes from L1 cache (this is not necessarily P_{peak})**

2. **I = Computational intensity ("work" per byte transferred) over the slowest data path utilized ("the bottleneck")**
   - Code balance $B_C = I^{-1}$

3. **b_S = Applicable peak bandwidth of the slowest data path utilized**

Expected performance:

$$P = \min\left(P_{\max}, \overset{[F/B]}{I} \cdot \overset{[B/s]}{b_S}\right)$$

[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)

Example: Estimate P_{max} of vector triad on SandyBridge

```
double  *A, *B, *C, *D;
for (int i=0; i<N; i++) {
   A[i] = B[i] + C[i] * D[i]
}
```

How many cycles to process one 64-byte cache line (one core)?

64byte = equivalent to 8 scalar iterations or **2** AVX vector iterations.

Cycle 1: load and ½ store and mult and add
Cycle 2: load and ½ store
Cycle 3: load

Answer: 6 cycles

Example: Estimate P_{max} of vector triad on SandyBridge

```
double  *A, *B, *C, *D;
for (int i=0; i<N; i++) {
   A[i] = B[i] + C[i] * D[i]
}
```

What is the performance in GFlops/s and the bandwidth in MBytes/s?

One AVX iteration (3 cycles) performs 4 x 2 = 8 flops.

(2.7 GHz / 3 cycles) * 4 updates * 2 flops/update = **7.2 GFlops/s**
4 GUPS/s * 4 words/update * 8byte/word = **128 GBytes/s**

P_{max} + bandwidth limitations: The vector triad

Example: Vector triad `A(:)=B(:)+C(:)*D(:)`
on a 2.7 GHz 8-core Sandy Bridge chip (AVX vectorized)

- b_S = **40 GB/s**
- B_c = (4+1) Words / 2 Flops = 2.5 W/F (including write allocate)

  → ***I*** **= 0.4 F/W = 0.05 F/B**

→ $I \cdot b_S$ = **2.0 GF/s (1.2 % of peak performance)**

- P_{peak} = **173 Gflop/s** (8 FP units x (4+4) Flops/cy x 2.7 GHz)
- P_{max} = 8 x 7.2 Gflop/s = **57.6 Gflop/s** (33% peak)

$$P = \min(P_{max}, I \cdot b_S) = \min(57.6, 2.0)\ \mathrm{GFlop/s}$$
$$= 2.0\ \mathrm{GFlop/s}$$

Exercise: Dense matrix-vector multiplication

```
do i=1,N
 do j=1,N
  c(i)=c(i)+A(j,i)*b(j)
 enddo
enddo
```

➡

```
do i=1,N
tmp = c(i)
 do j=1,N
  tmp = tmp + A(j,i)* b(j)
 enddo
 c(i) = tmp
enddo
```

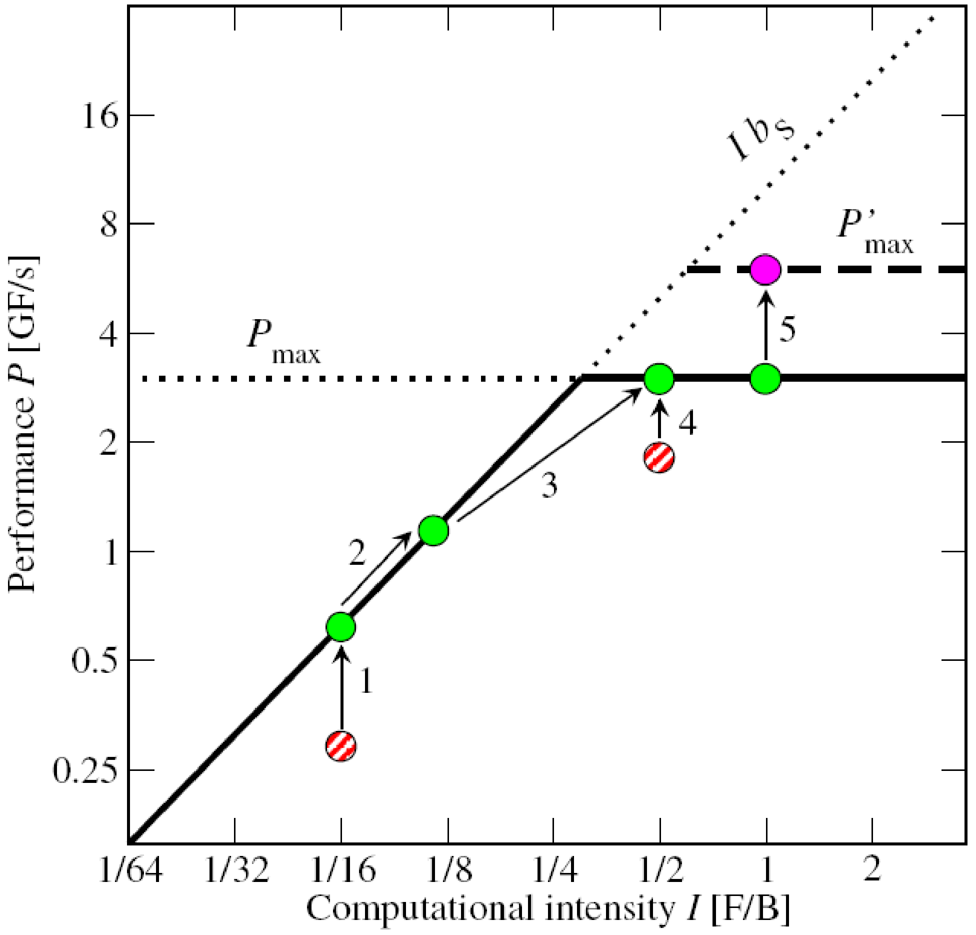
- **Assume N ≈ 5000**
- **Applicable peak performance?**
- **Relevant data path?**
- **Computational Intensity?**

Assumptions for the Roofline Model

- **The roofline formalism is based on some (crucial) assumptions:**
  - There is a clear concept of "work" vs. "traffic"
    - "work" = flops, updates, iterations…
    - "traffic" = required data to do "work"
  - Attainable bandwidth of code = input parameter! Determine effective bandwidth via simple streaming benchmarks to model more complex kernels and applications
  - Data transfer and core execution overlap perfectly!
  - Slowest data path is modeled only; all others are assumed to be infinitely fast
  - If data transfer is the limiting factor, the bandwidth of the slowest data path can be utilized to 100% ("saturation")
  - Latency effects are ignored, i.e. perfect streaming mode

Typical code optimizations in the Roofline Model

1. Hit the BW bottleneck by good serial code
2. Increase intensity to make better use of BW bottleneck
3. Increase intensity and go from memory-bound to core-bound
4. Hit the core bottleneck by good serial code
5. Shift P_{max} by accessing additional hardware features or using a different algorithm/implementation



Case study:
A 3D Jacobi smoother

The basics in two dimensions

Roofline performance analysis and modeling

A Jacobi smoother

- **Laplace equation in 2D:** $\Delta\Phi = 0$

- **Solve with Dirichlet boundary conditions using Jacobi iteration scheme:**

```
double precision, dimension(0:imax+1,0:kmax+1,0:1) :: phi
integer :: t0,t1
t0 = 0 ; t1 = 1
do it = 1,itmax     ! choose suitable number of sweeps
  do k = 1,kmax
    do i = 1,imax
      ! four flops, one store, four loads
      phi(i,k,t1) = (  phi(i+1,k,t0) + phi(i-1,k,t0)
                     + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25
    enddo
  enddo
  ! swap arrays
  i = t0 ; t0=t1 ; t1=i
enddo
```

Re-use when computing `phi(i+2,k,t1)`

WRITE ALLOCATE:
LD + ST `phi(i,k,t1)`

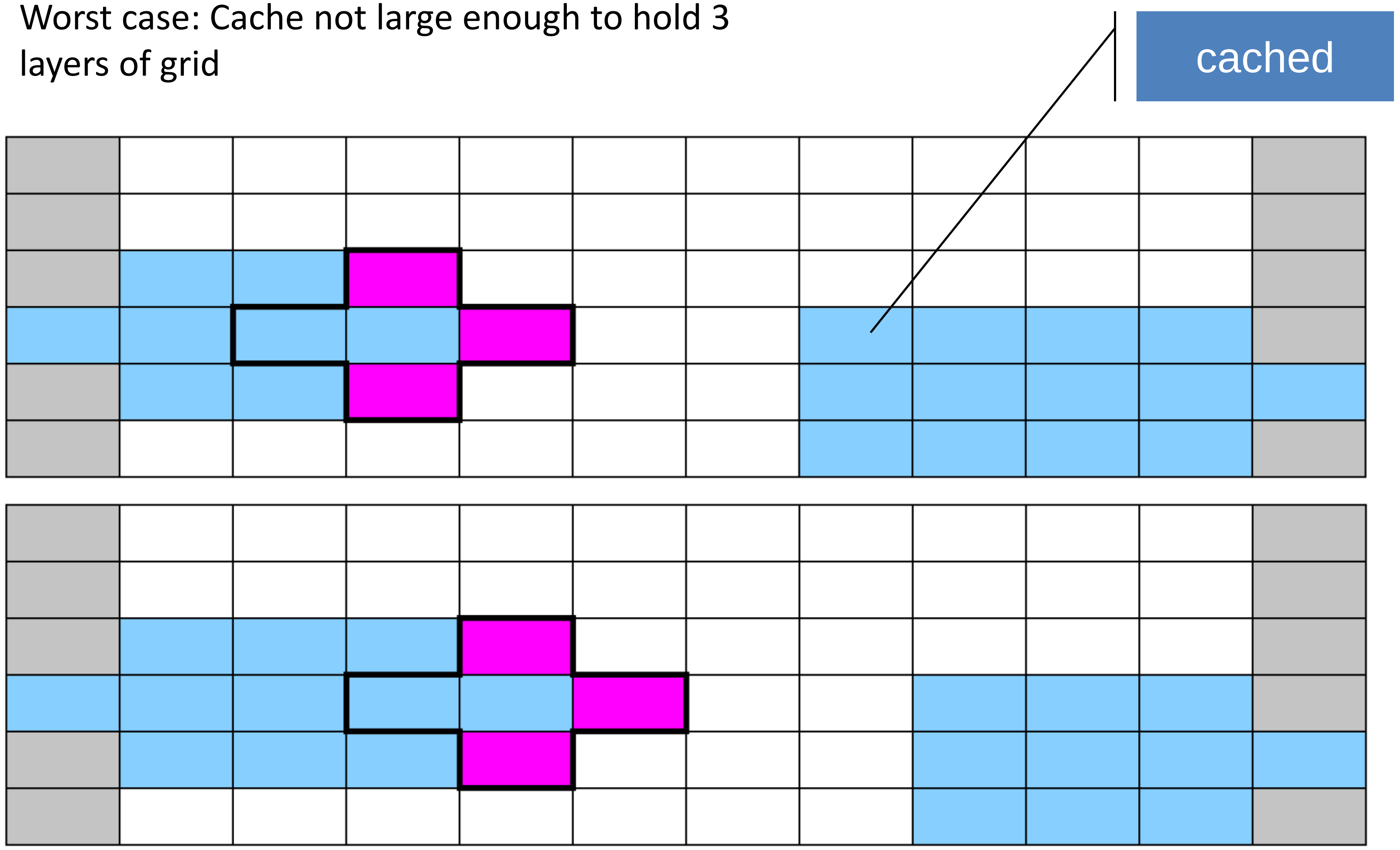
Naive balance (incl. write allocate):

`phi(:,:,t0)`: 3 LD +
`phi(:,:,t1)`: 1 ST+ 1LD

→ B_C = 5 W / 4 FLOPs = 10 B/F (= 40 B/LUP)

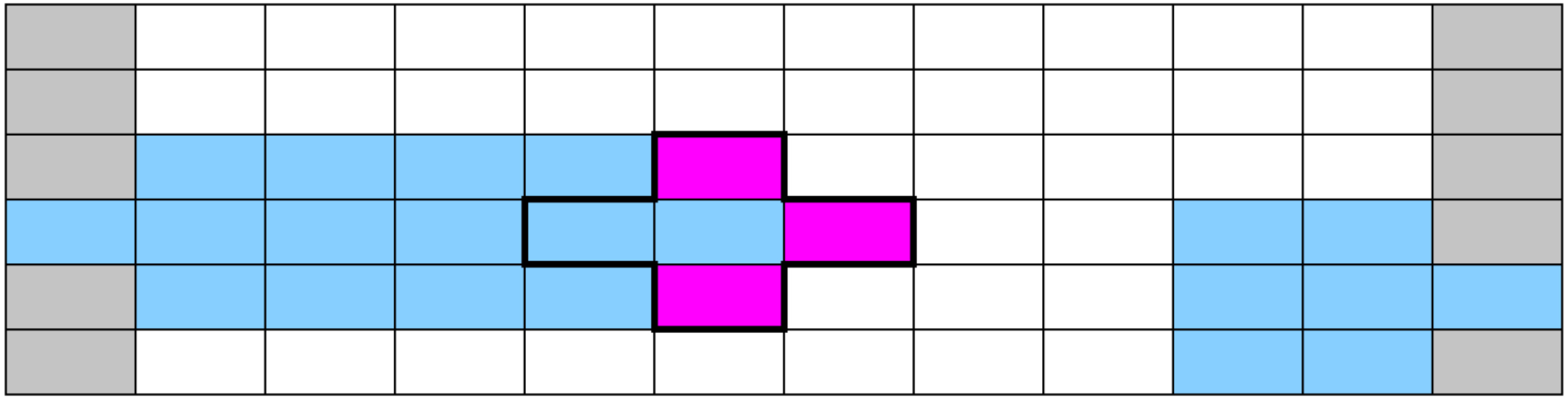
Analyzing the data flow

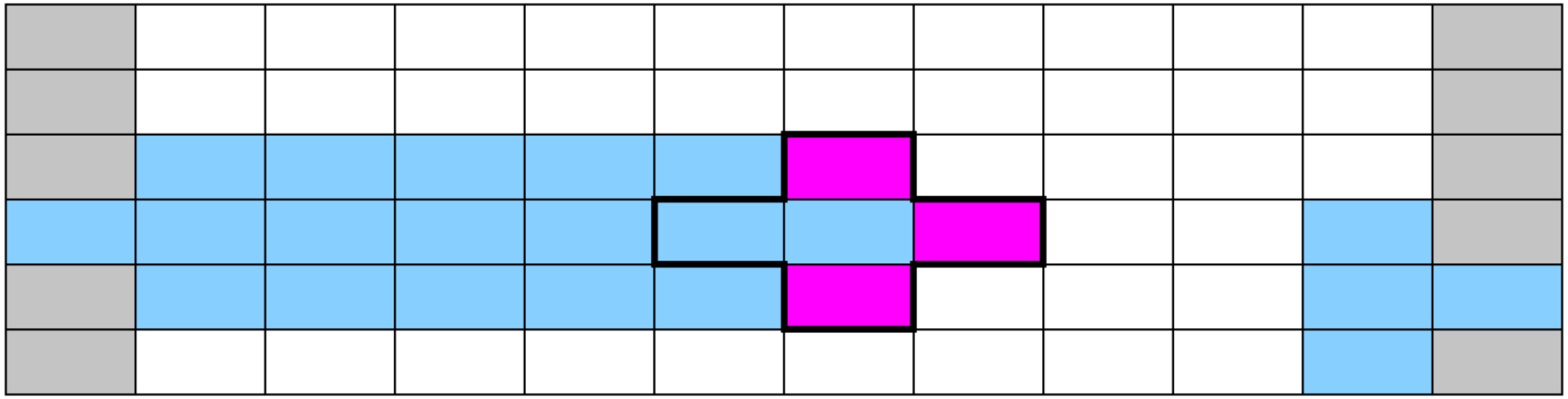
Worst case: Cache not large enough to hold 3 layers of grid



Analyzing the data flow

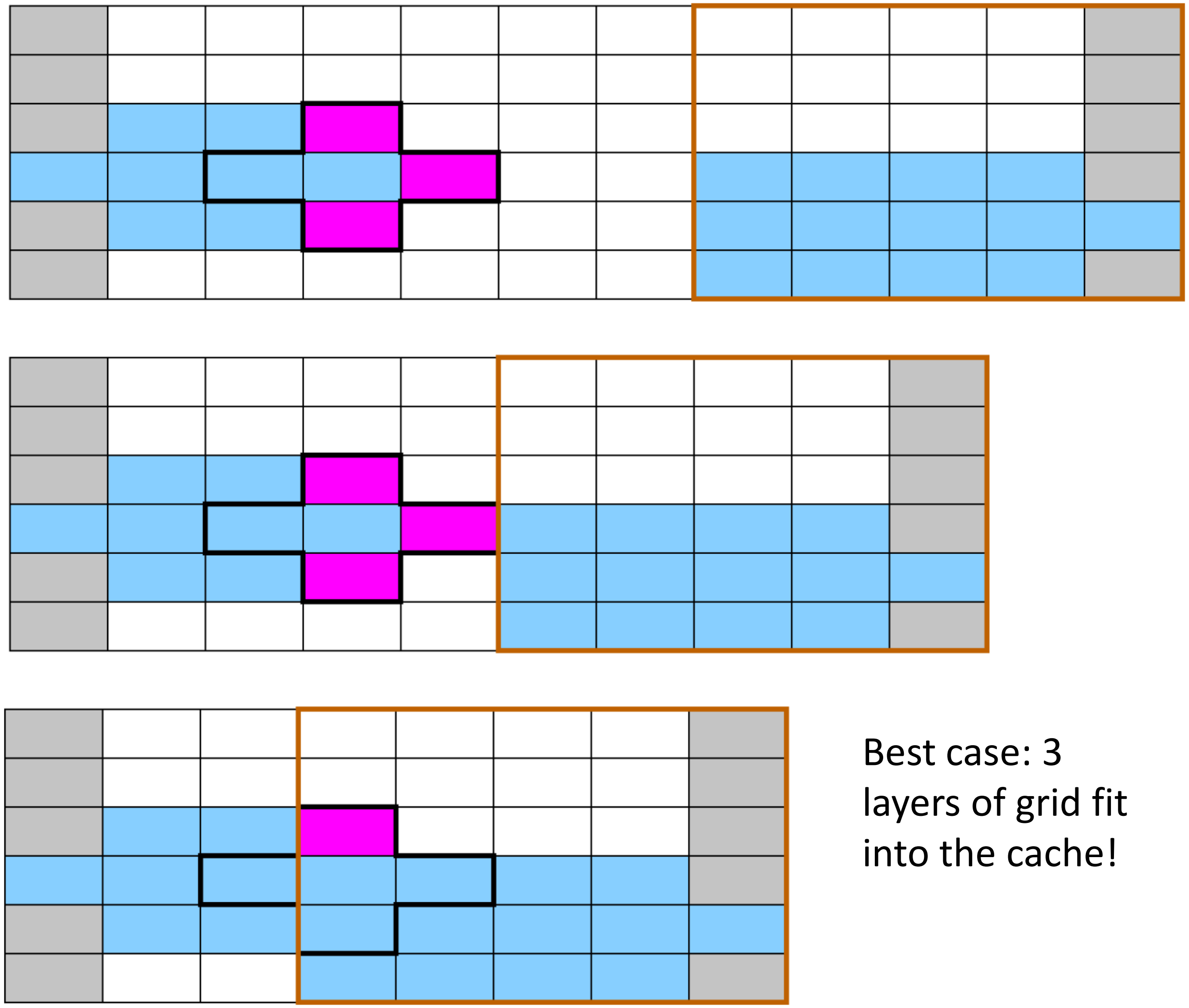
Worst case: Cache not large enough to hold 3 layers of grid





Analyzing the data flow

Making the inner lop dimension successively smaller



Best case: 3 layers of grid fit into the cache!

Balance metric: 2 D Jacobi

- Modern cache subsystems may further reduce memory traffic
  - → "layer conditions"

If cache is large enough to hold at least 3 rows:
Each `phi(:,:,t0)` is loaded once from main memory and re-used 3 times from cache:

`phi(:,:,t0)`: 1 LD + `phi(:,:,t1)`: 1 ST+ 1LD

→B_C = 3 W / 4 F = 24 B/LUP

If cache is too small :
`phi(:,:,t0)`: 3 LD + `phi(:,:,t1)`: 1 ST+ 1LD
→B_C = 5 W / 4 F = 40 B/LUP

2D → 3D

- **3D sweep:**

Layer condition:
`nthreads*3*jmax*imax*8B < CS/2`

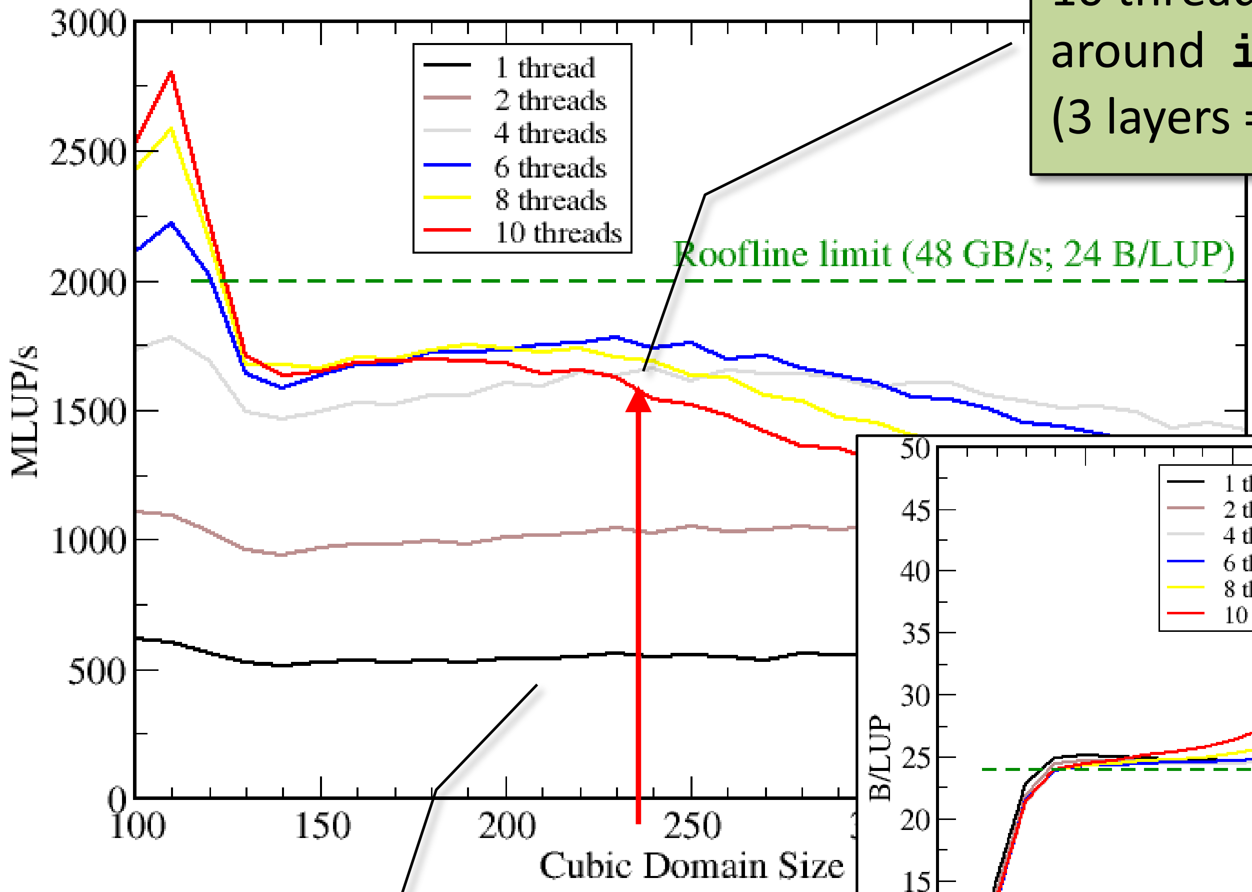
```
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      phi(i,j,k,t1) = 1/6. *(phi(i-1,j,k,t0)+phi(i+1,j,k,t0) &
                           + phi(i,j-1,k,t0)+phi(i,j+1,k,t0) &
                           + phi(i,j,k-1,t0)+phi(i,j,k+1,t0))
    enddo
  enddo
enddo
```

- **Best case balance:** 1 LD `phi(i,j,k+1,t0)`
  1 ST + 1 write allocate `phi(i,j,k,t1)`
  6 flops

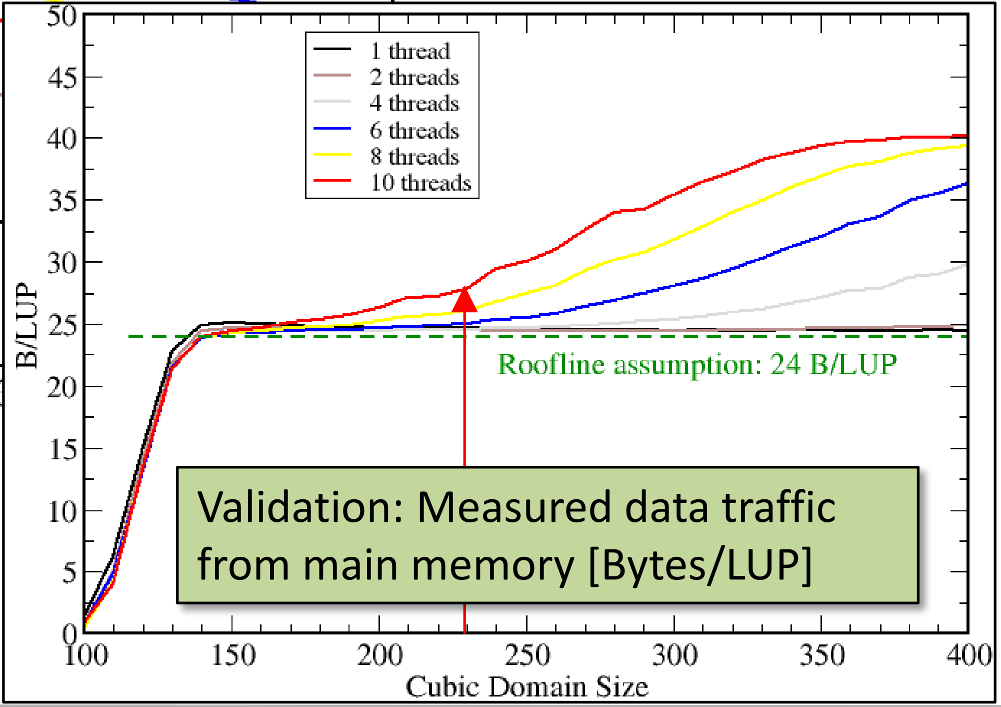
  → B_C = 0.5 W/F (24 B/LUP)

- **No 3-layer condition but 3 rows fit:** B_C = 5/6 W/F (40 B/LUP)
- **Worst case (3 rows do not fit):** B_C = 7/6 W/F (56 B/LUP)

Jacobi Stencil – Observed performance vs. problem size



10 threads: performs starts to drop around **`imax=230`**
(3 layers = 13 MB, CS = 25 MB)



Validation: Measured data traffic from main memory [Bytes/LUP]

1 thread: Layer condition OK – but can not saturate bandwidth

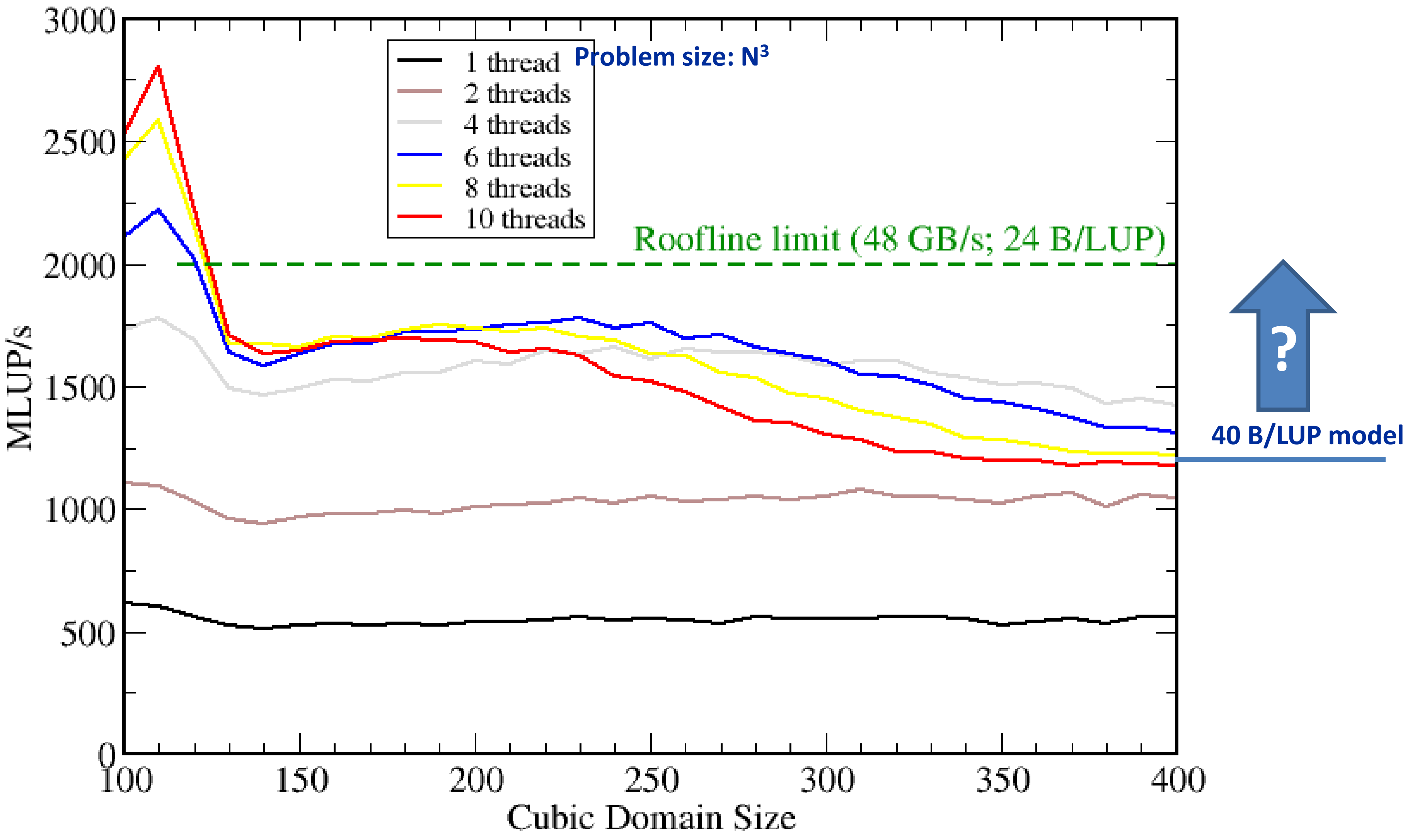
Conclusions from the Jacobi example

- **We have made sense of the memory-bound performance vs. problem size**
  - "Layer conditions" lead to predictions of code balance
  - Achievable memory bandwidth is input parameter
- **The model works only if the bandwidth is "saturated"**
  - In-cache modeling is more involved
- **Hardware performance counters (likwid-perfctr)**
  - Traffic volume per LUP measured using cache lines loaded/evicted from/to memory
  - Used for model validation
- **Optimization == reducing the code balance by code transformations**
  - See below

Data access optimizations

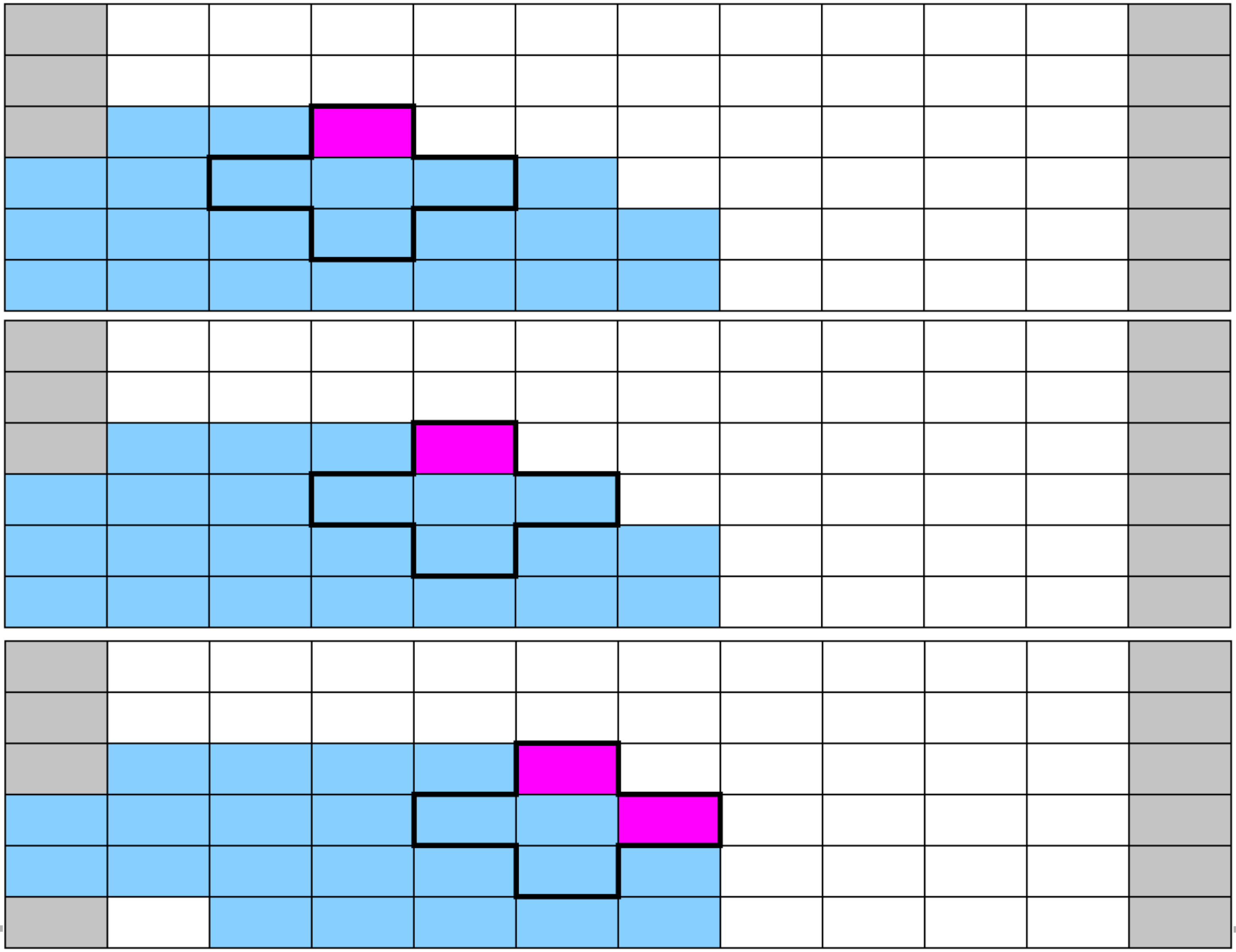
Case study: Optimizing the 3D Jacobi solver

How can we re-establish the layer condition?



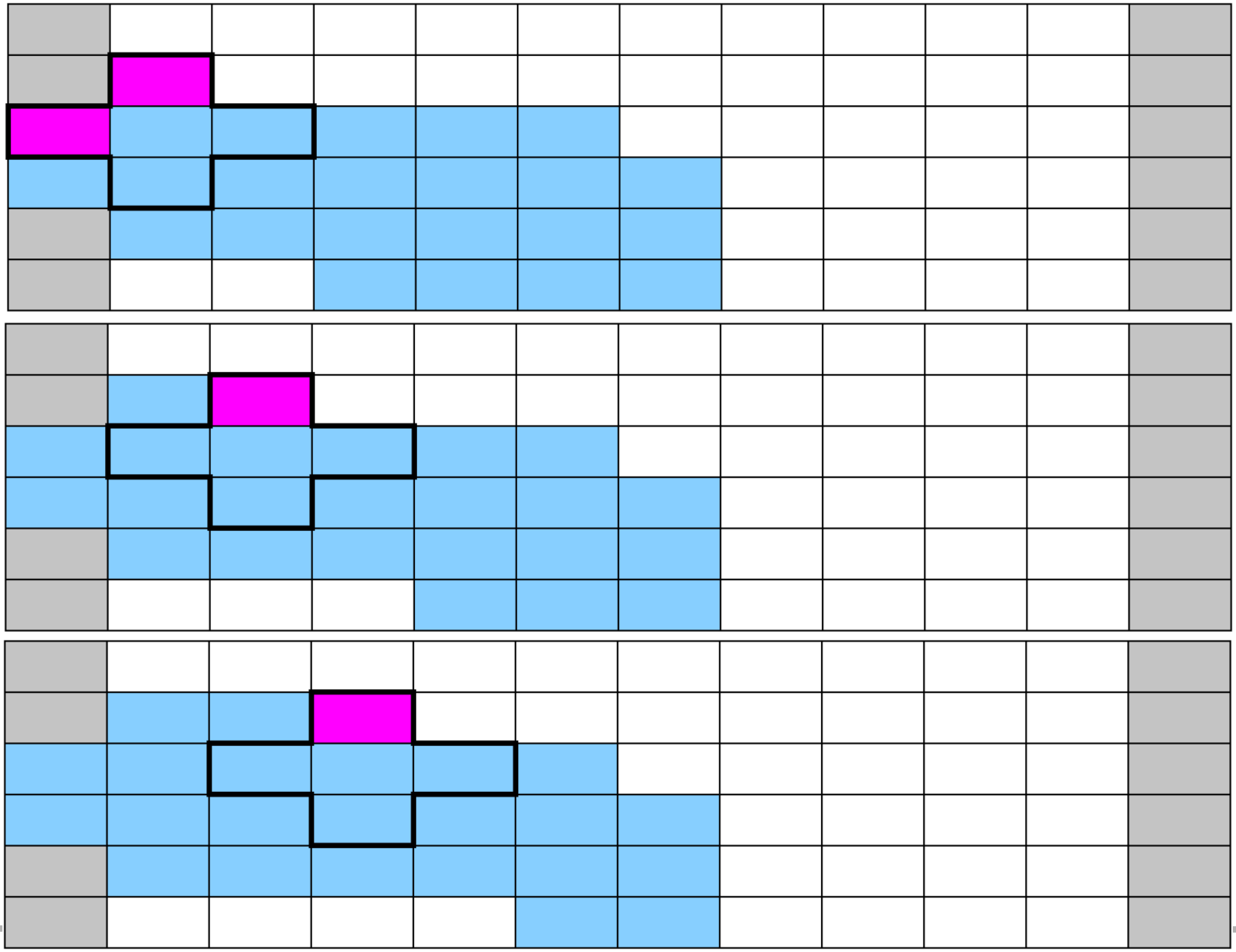
Enforcing the layer condition by blocking

Inner loop block size = 5



Enforcing the layer condition by blocking

Inner loop block size = 5



Jacobi Stencil – simple spatial blocking

```
do jb=1,jmax,jblock ! Assume jmax is multiple of jblock

!$OMP PARALLEL DO SCHEDULE(STATIC)
  do k=1,kmax
    do j=jb,(jb+jblock-1) !   Loop length jblock
      do i=1,imax
        phi(i,j,k,t1) = (phi(i-1,j,k,t0)+phi(i+1,j,k,t0) &
                       + phi(i,j-1,k,t0)+phi(i,j+1,k,t0) &
                       + phi(i,j,k-1,t0) +phi(i,j,k+1,t0)) * 1/6.d0
      enddo
    enddo
  enddo

enddo
```

"Layer condition" (j-Blocking))

`nthreads*3*jblock*imax*8B < CS/2`

Ensure layer condition by choosing `jblock` appropriately (cubic domains):

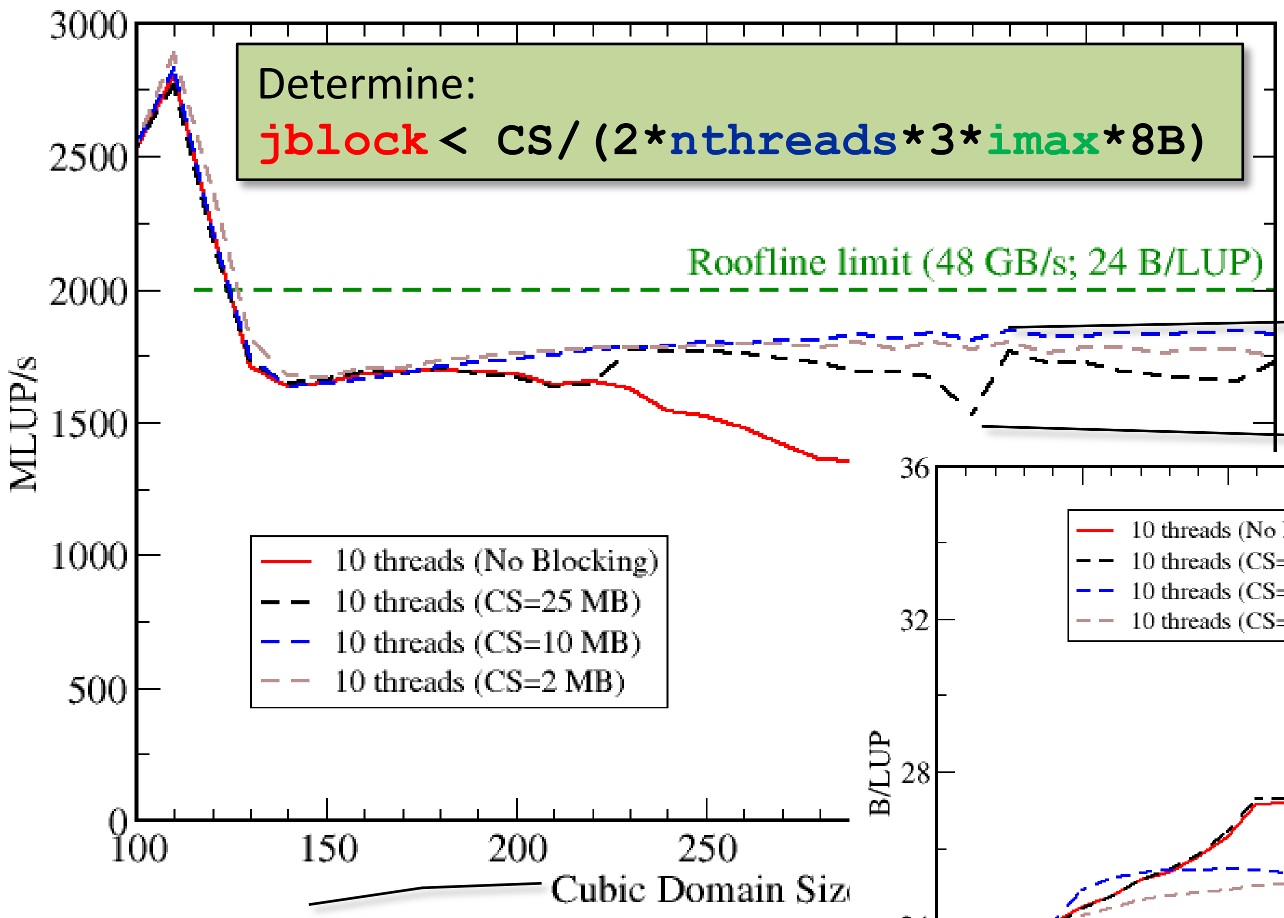
`jblock < CS/(imax * nthreads * 48 B )`

Test system: Intel Xeon E5-2690 v2 (10 cores / 3 GHz)

b_S = 48 GB/s , CS = 25 MB (L3)

→ P = 2000 MLUP/s

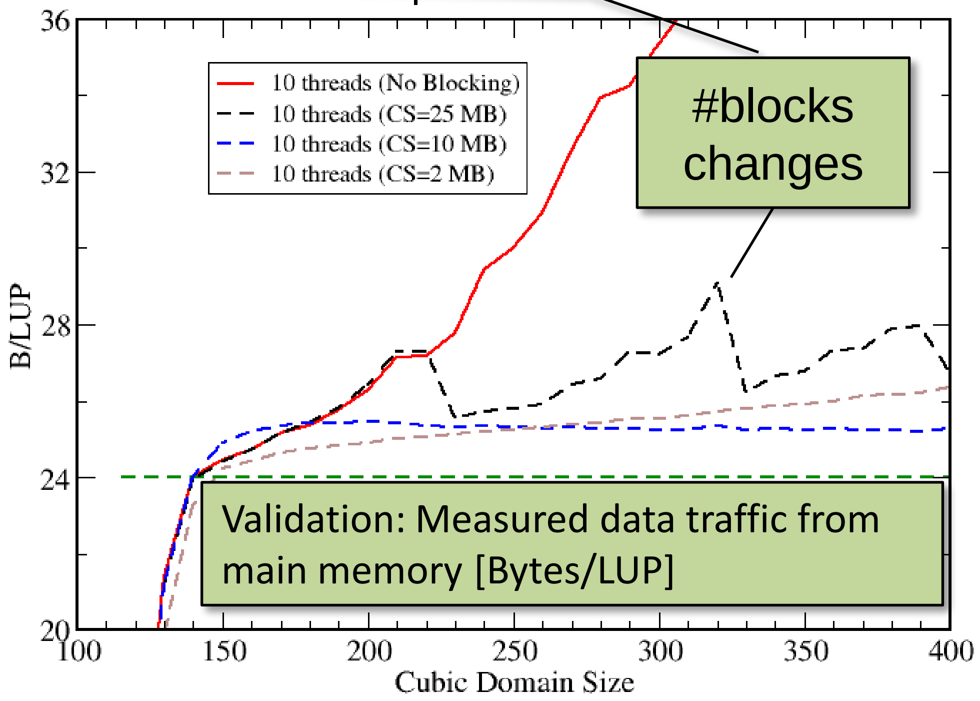
Spatial blocking



Determine:
`jblock < CS/(2*nthreads*3*imax*8B)`

CS=10 MB:
~ 90% of Roofline limit

`imax = jmax = kmax`



#blocks changes

Validation: Measured data traffic from main memory [Bytes/LUP]

Conclusions from the Jacobi optimization example

- **“What part of the data comes from where” is a crucial question**

- **Avoiding slow data paths == re-establishing the most favorable layer condition**

- **Improved code showed the speedup predicted by the model**

- **Optimal blocking factor can be estimated**
  - Be guided by the cache size the layer condition
  - No need for exhaustive scan of “optimization space”

- **Non-temporal stores avoid the write-allocate and thus reduce memory traffic**
  - But they may come at a price

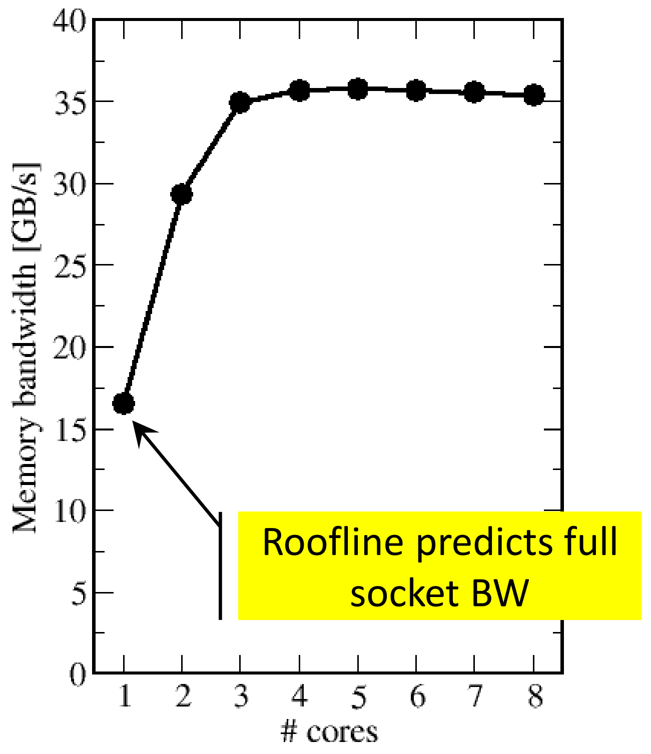
Shortcomings of the roofline model

- **Saturation effects in multicore chips are not explained**
  - Reason: “saturation assumption”
  - Cache line transfers and core execution do sometimes not overlap perfectly
  - Only increased “pressure” on the memory interface can saturate the bus
    → need more cores!

- **ECM model gives more insight**

G. Hager, J. Treibig, J. Habich, and G. Wellein: Exploring performance and power properties of modern multicore chips via simple machine models. Accepted for publication in Concurrency and Computation: Practice and Experience. Preprint: arXiv:1208.2908

`A(:)=B(:)+C(:)*D(:)`



The Execution-Cache-Memory (ECM) model

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). Preprint: arXiv:1208.2908

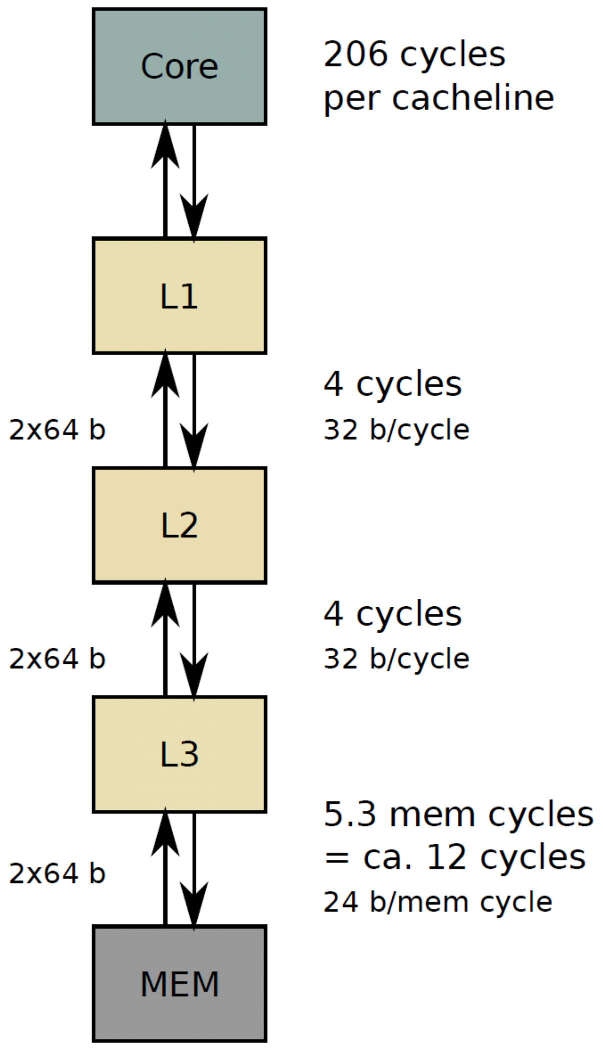
J. Treibig and G. Hager: *Introducing a Performance Model for Bandwidth-Limited Loop Kernels.* Proc. PPAM 2009, Lecture Notes in Computer Science Volume 6067, 615-624 (2010). DOI: 10.1007/978-3-642-14390-8_64. Preprint: arXiv:0905.0792

ECM Model

- **ECM = “Execution-Cache-Memory”**

- **Assumptions:**
- **Single-core execution time is composed of**
  1. In-core execution
  2. Data transfers in the memory hierarchy
  - **Data transfers may or may not overlap with each other or with in-core execution**
  - **Scaling is linear until the relevant bottleneck is reached**

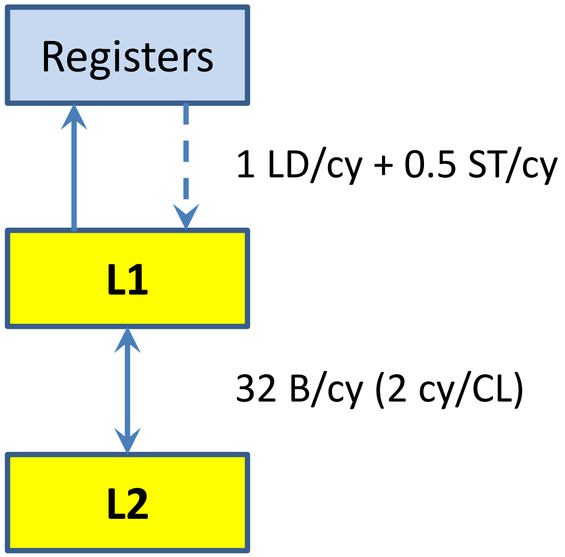
  - **Input:**
  - **Same as for Roofline**
  - **+ data transfer times in hierarchy**



Example: Schönauer Vector Triad in L2 cache

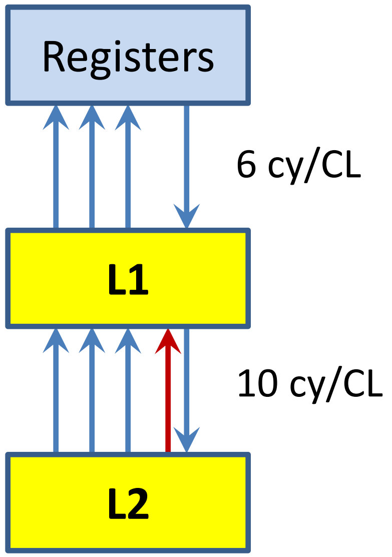
- REPEAT[`A(:) = B(:) + C(:) * D(:)`] @ double precision
- Analysis for Sandy Bridge core w/ AVX (unit of work: 1 cache line)

Machine characteristics:



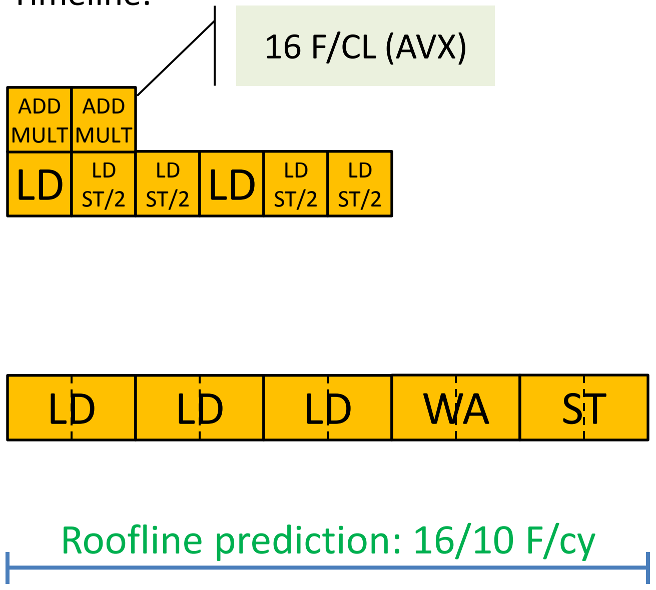
Arithmetic:
1 ADD/cy+ 1 MULT/cy

Triad analysis (per CL):



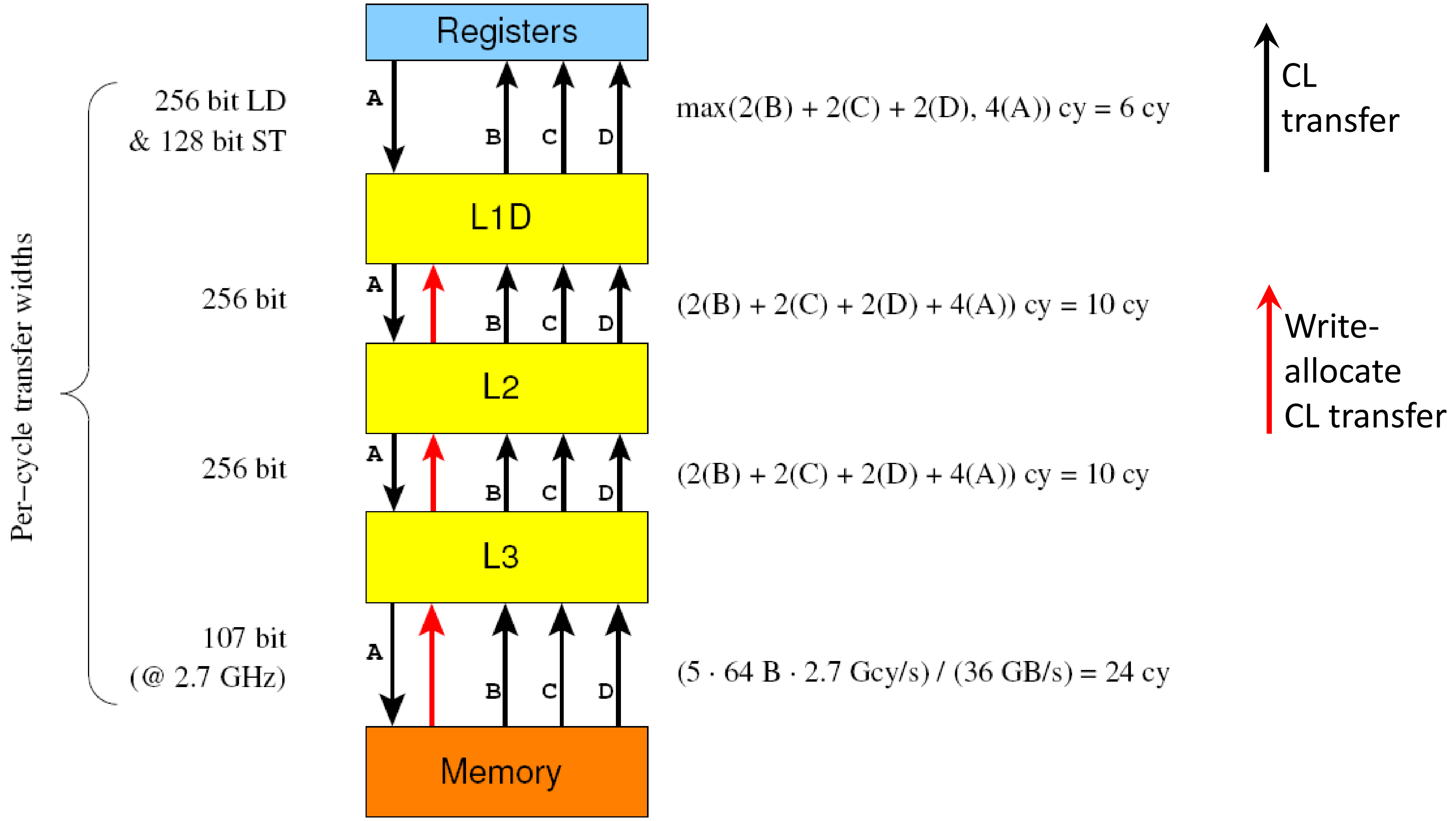
Arithmetic:
AVX: 2 cy/CL

Timeline:

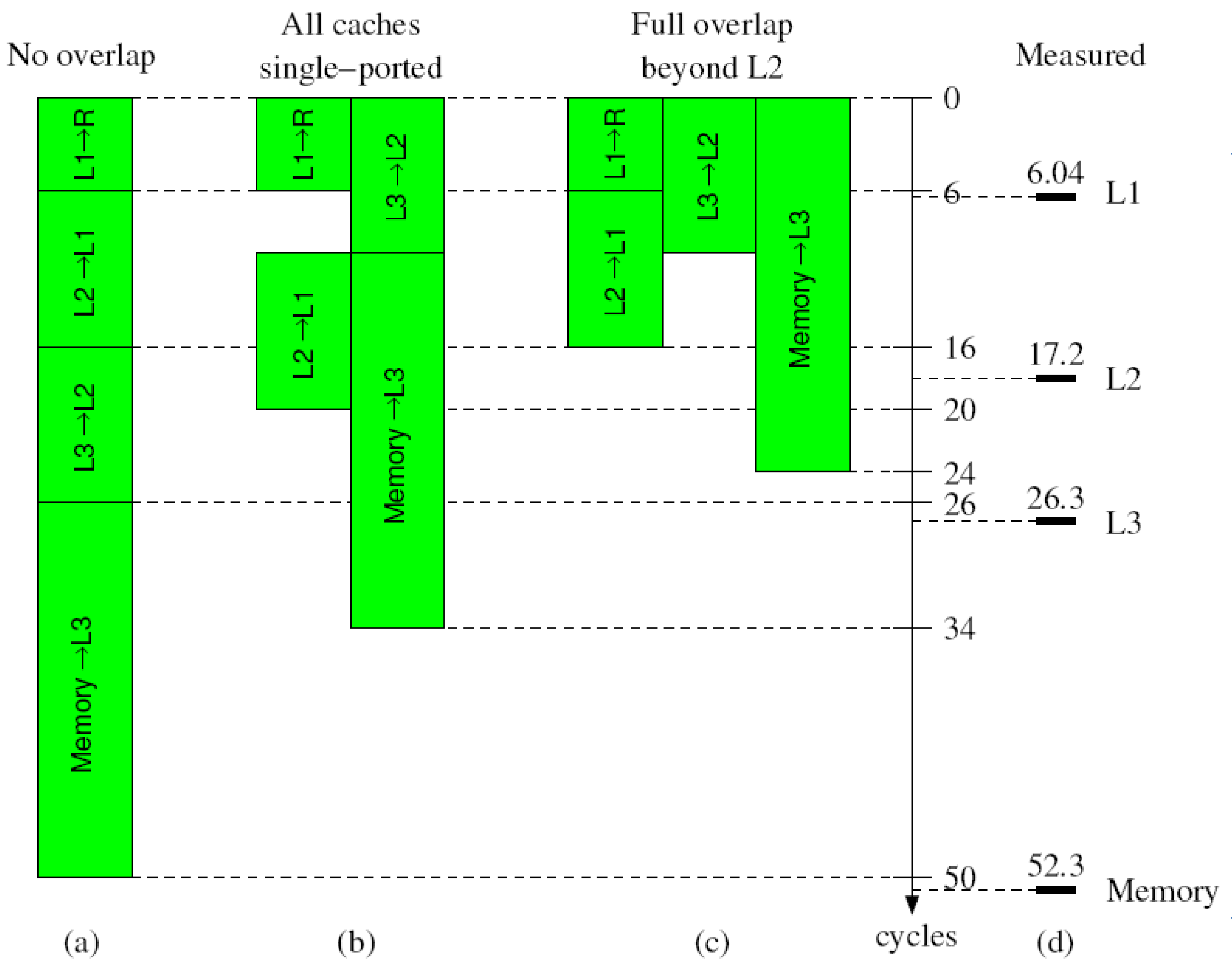


Measurement: 16F / ≈17cy

Example: ECM model for Schönauer Vector Triad `A(:)=B(:)+C(:)*D(:)` on a Sandy Bridge Core with AVX


Registers
L1D
L2
L3
Memory
A
B
C
D
Per–cycle transfer widths
256 bit LD & 128 bit ST
256 bit
256 bit
107 bit (@ 2.7 GHz)
max(2(B) + 2(C) + 2(D), 4(A)) cy = 6 cy
(2(B) + 2(C) + 2(D) + 4(A)) cy = 10 cy
(2(B) + 2(C) + 2(D) + 4(A)) cy = 10 cy
(5 · 64 B · 2.7 Gcy/s) / (36 GB/s) = 24 cy
CL transfer
Write-allocate CL transfer

Full vs. partial vs. no overlap



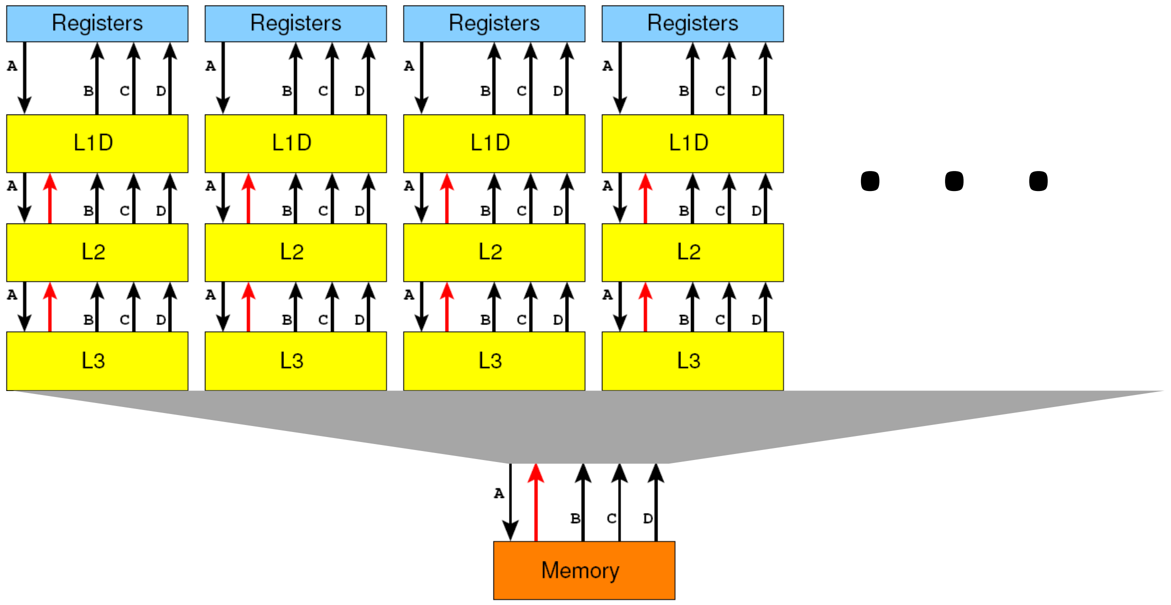
Results suggest no overlap!

Multicore scaling in the ECM model

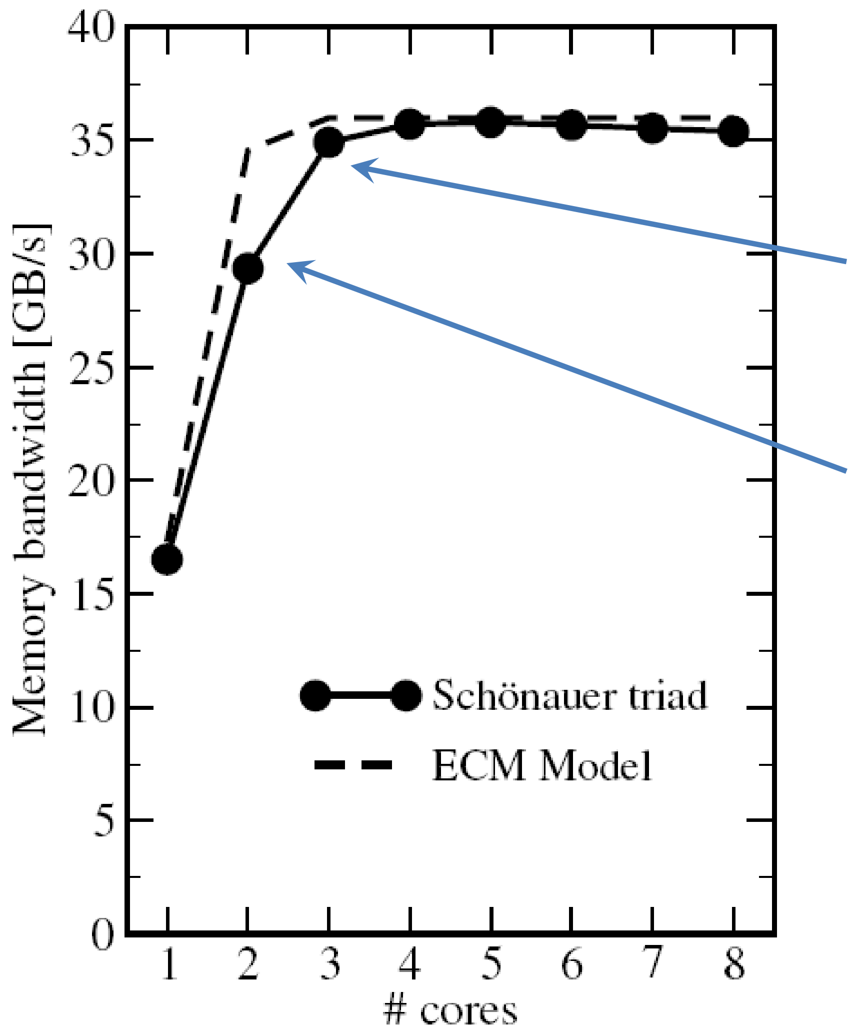
- **Identify relevant bandwidth bottlenecks**
  - L3 cache
  - Memory interface
- **Scale single-thread performance until first bottleneck is hit:**

$$P(t) = \min(tP_0, I \cdot b_S)$$

Example: Scalable L3 on Sandy Bridge



ECM prediction vs. measurements for `A(:)=B(:)+C(:)*D(:)` on a Sandy Bridge socket (no-overlap assumption)



Model: Scales until saturation sets in

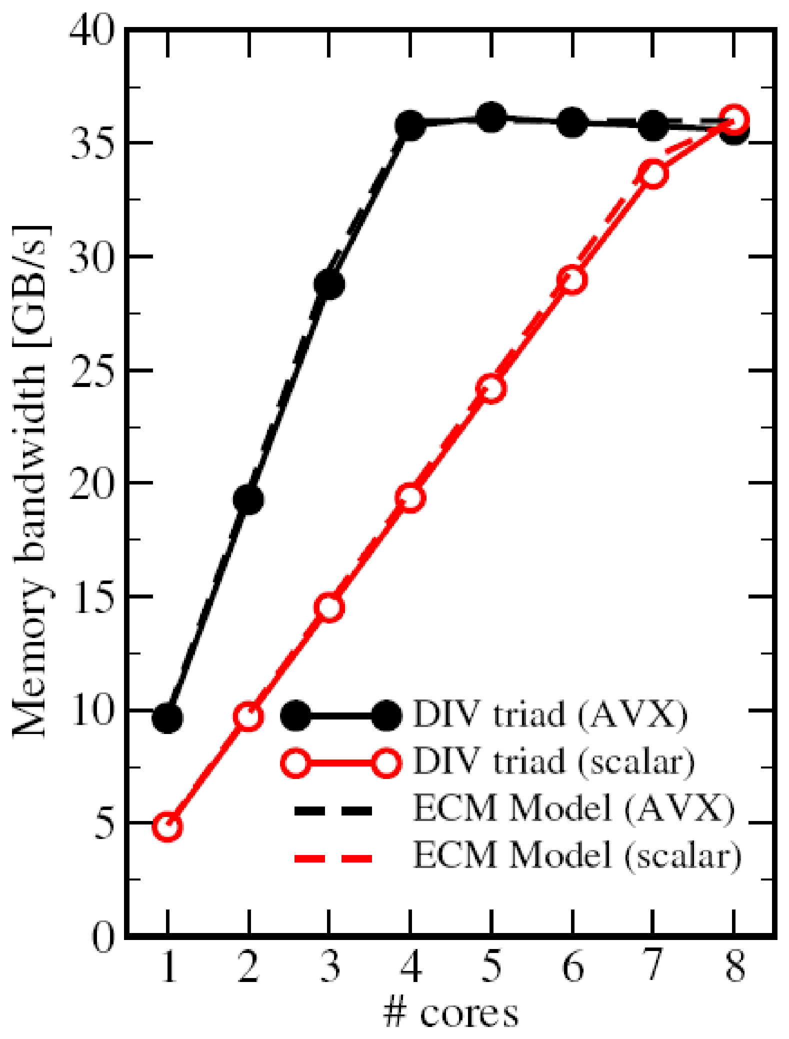
Saturation point (# cores) well predicted

Measurement: scaling not perfect

Caveat: This is specific for this architecture and this benchmark!

Check: Use "overlappable" kernel code

ECM prediction vs. measurements for `A(:)=B(:)+C(:)/D(:)` on a Sandy Bridge socket (full overlap assumption)



In-core execution is dominated by divide operation

(44 cycles with AVX, 22 scalar)

→Almost perfect agreement with ECM model

General observation:

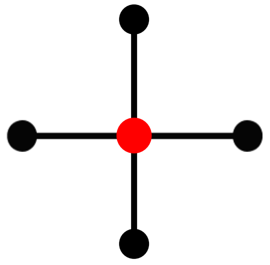
- If the L1 cache is 100% occupied by LD/ST, there is no overlap throughout the hierarchy
- If there is “slack” at the L1, there is some overlap in the hierarchy

Performance Modeling of Stencil Codes

Applying the ECM model to a 2D Jacobi smoother

(H. Stengel, RRZE)

Example 1: 2D Jacobi in double precision with SSE2 on Sandy Bridge



Example 1: 2D Jacobi in DP with SSE2 on SNB

```c
// Jacobi 2D line update
for(int j=start; j<end; j++){
     t1[i][j]= ( t0[i-1][j]   +
                 t0[i+1][j]   +
                 t0[i][j-1]   +
                 t0[i][j+1] ) * 0.25;
}
```

```
movups     (%rbp,%r15,8), %xmm1
movups     16(%rbp,%r15,8), %xmm3
movups     32(%rbp,%r15,8), %xmm5
movups     48(%rbp,%r15,8), %xmm7
addpd      (%r9,%r15,8), %xmm1
addpd      16(%r9,%r15,8), %xmm3
addpd      32(%r9,%r15,8), %xmm5
addpd      48(%r9,%r15,8), %xmm7
addpd      -8(%r10,%r15,8), %xmm1
movups     8(%r10,%r15,8), %xmm2
movups     24(%r10,%r15,8), %xmm4
movups     40(%r10,%r15,8), %xmm6
addpd      %xmm2, %xmm3
addpd      %xmm4, %xmm5
addpd      %xmm6, %xmm7
addpd      %xmm2, %xmm1
addpd      %xmm4, %xmm3
addpd      %xmm6, %xmm5
addpd      56(%r10,%r15,8), %xmm7
mulpd      %xmm0, %xmm1
mulpd      %xmm0, %xmm3
mulpd      %xmm0, %xmm5
mulpd      %xmm0, %xmm7
movups     %xmm1, (%r11,%r15,8)
movups     %xmm3, 16(%r11,%r15,8)
movups     %xmm5, 32(%r11,%r15,8)
movups     %xmm7, 48(%r11,%r15,8)
```

4-way unrolling
→ 8 LUP / iteration

Instruction count
- **13 LOAD**
- **4 STORE**
- **12 ADD**
- **4 MUL**

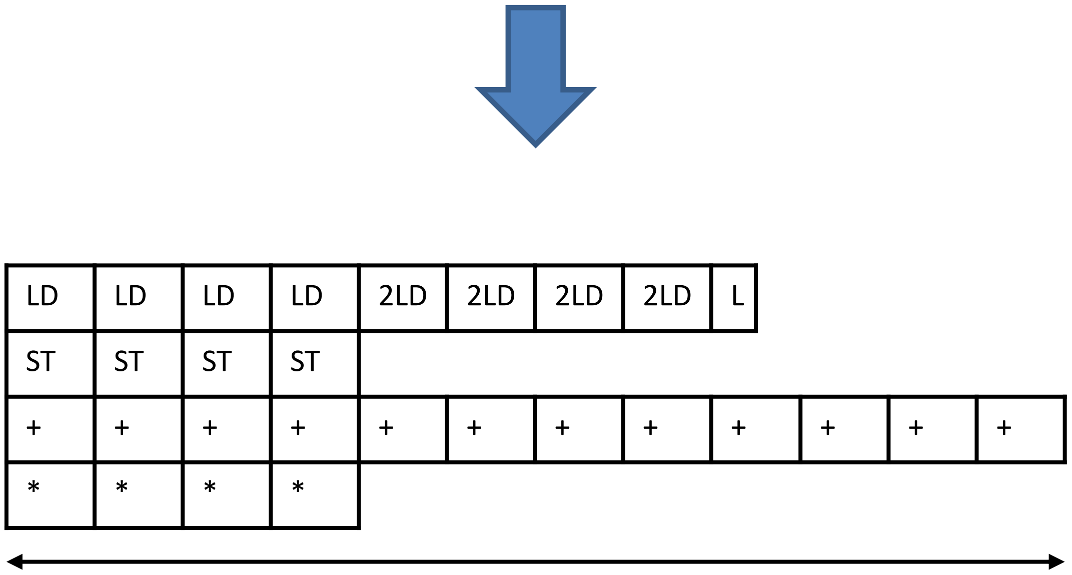
Example 1: 2D Jacobi in DP with SSE2 on SNB

<u>**Processor characteristics**</u>
(SSE instructions per cycle)

- **2 LOAD || (1 LOAD + 1 STORE)**
- **1 ADD**
- **1 MUL**

<u>**Code characteristics**</u>
(SSE instructions per iteration)

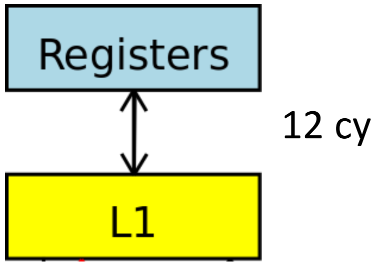
- **13 LOAD**
- **4 STORE**
- **12 ADD**
- **4 MUL**



<u>**core execution:**</u>
12 cy

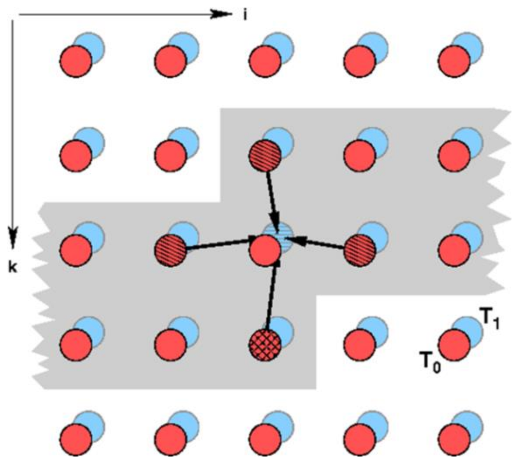
Example 1: 2D Jacobi in DP with SSE2 on SNB

- **Situation 1: Data set fits into L1 cache**
  - ECM prediction:
    (8 LUP / 12 cy) * 3.5 GHz = 2.3 GLUP/s
  - Measurement: 2.2 GLUP/s

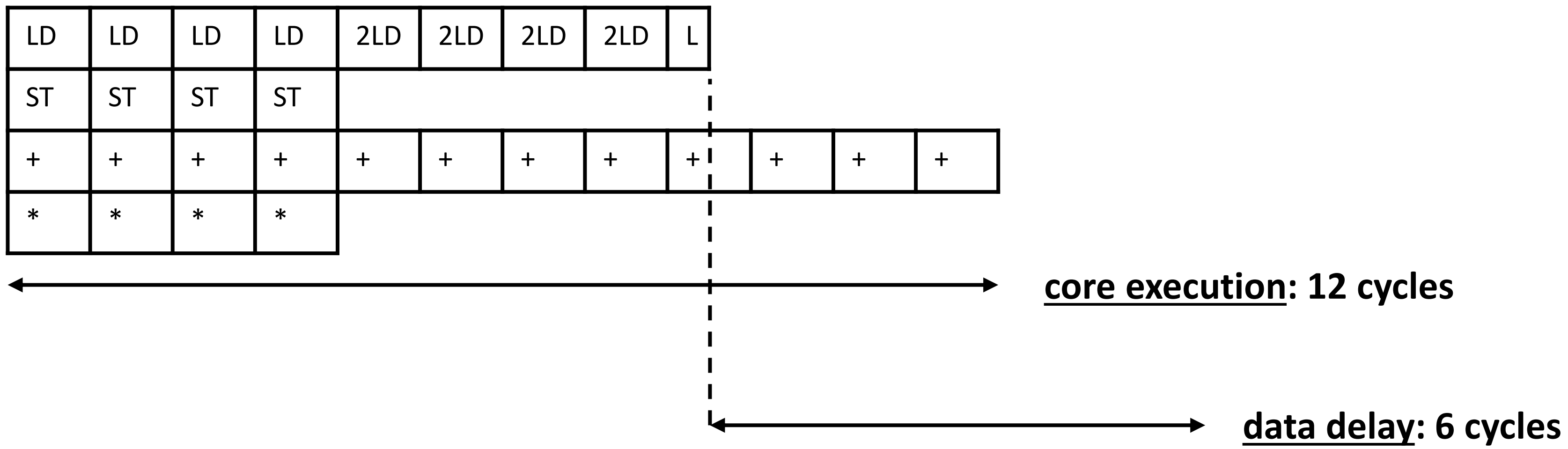


- **Situation 2: Data set fits into L2 cache (not into L1)**
  - 3 additional transfer streams from L2 to L1 (data delay)
  - ECM prediction:
    (8 LUP / (12+6) cy) * 3.5 GHz = 1.5 GLUP/s
  - Measurement: 1.9 GLUP/s

Overlap?

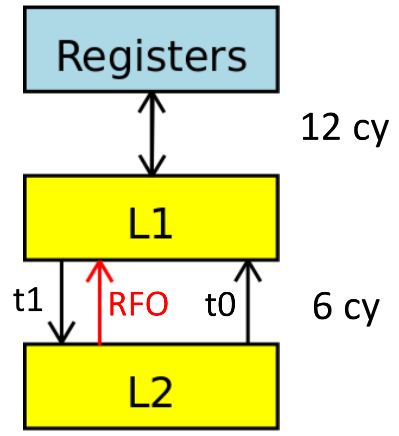


Example 1: 2D Jacobi in DP with SSE2 on SNB



L1 „single ported“
→ no overlap during LD/ST

- ECM prediction w/ overlap:
  (8 LUP / (8.5+6) cy) * 3.5 GHz = 1.9 GLUP/s
- Measurement: 1.9 GLUP/s



“If the model fails, we learn something”

Conclusions

- **Performance models help us understand more about**
  - Interaction of software with hardware
  - Optimization opportunities

- **Roofline Model**
  - "Simple" bottleneck analysis
  - Good for "saturated" situations (full chip)
  - Benefit of blocking / traffic saving optimizations can be predicted
  - Shortcomings: Single data bottleneck, perfect overlap assumption
    → multicore scaling cannot be modeled

- **ECM Model**
  - Multiple bottlenecks: Execution, Caches, Memory
  - 1st shot: Assume no overlap in hierarchy
  - Good single-core predictions, converges to Roofline in saturated case