ERLANGEN REGIONAL
COMPUTING CENTER [RRZE]

Detecting performance limiting factors with hardware monitoring

Thomas Gruber, HPC group, FAU Erlangen

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Agenda

- What is performance?
- What are limiting factors?
- Introduction to LIKWID
- Detection
  - Bad vectorization
  - Load imbalance
  - False sharing in cache hierarchy

Motivation

Motivation
HPC == *Computing on the edge*

- Execution limited by bottlenecks caused by
  - Hardware system limitations
  - Software runtime behavior
  - User code
- (Best) Approach in HPC:
  - Keep user code slim (main logic, glue code & library calls)
  - Use optimized software runtime and libraries
  - Hit hardware bottleneck

Of people who know what they are doing!

What is performance?

How to measure performance?

Performance
What is a good measure/metric?

My vector update code runs at 2,000 MFLOP/s on a 2GHz processor!

```
for(i=0; i<n; i++) {
    a[i]= 3.0*c0+c1*c2+c3*c4*a[i]-1.d0*a[i];
}
```

#FLOPS = 8 * n

Same execution time but…

- #FLOP = 2* n + 5

```
d0 = 3.0*c0+c1*c2;
d1 = c3*c4-1.d0;

for(i=0; i<n; i++) {
    a[i]= d0 + d1*a[i];
}
```

… but my MFlop/s rate is only ¼!

Performance
What is a good measure/metric?

- **Performance = WORK / TIME**
- **“Pure” metrics – basic choices for “WORK”**
  - Iterations: Total number of loop iterations performed
  - MFLOP: Millions of Floating Point Operations
  - MIPS: Millions of Instructions
  - Lattice Site/ Cell / Particle Updates: number of sites/cells/particles to be updated/computed
  - Physical simulation time: Physical time (e.g. nanosenconds) a system is propagated

Performance
What is a good measure/metric?

- Simplest performance metric (“Bestseller”): 1 / TIME
  - **Measures time to solution**
  - **Carefully specify the “problem” you solved!**
  - **Best metric thinkable, but not intuitive in all**
- LINUX / UNIX command `time`

Problem:
Which TIME?

Always use one dedicated source for time measurements (e.g. one node)

```
>time ./test.x
>34.650u 0.612s 0:35.26 99.9%
>time ./testwIO.x
>33.802u 0.608s 0:43.64 78.8%
 user     sys     elapsed ratio
  cpu time        time   (u+s)/e
```

Stay away from CPU time – it‘s evil! No I/O!

This is the time (walltime) you wait for the result!

Performance Measurement
Best Practices

- **Preparation**
  - Reliable timing/timer granularity
    (Minimum time which can be measured?)
  - Document code generation (Flags, Compiler Version)
  - Document system state (Clock, Turbo mode, Memory, Caches)
  - Consider to automate runs with a script (Shell, python, perl)
    if it works manually

Performance Measurement
Best Practices

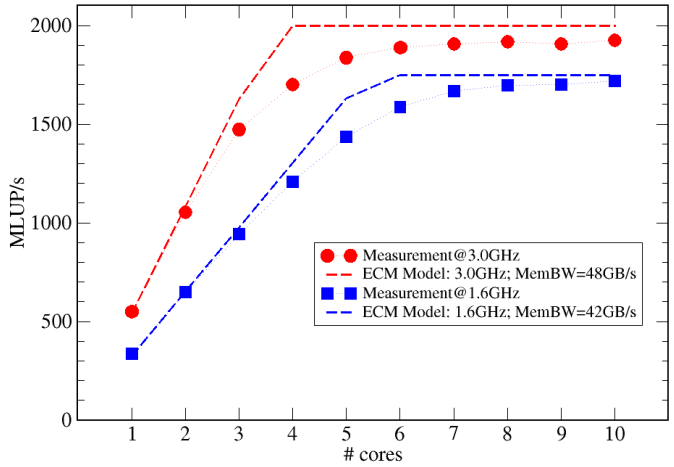
- **Doing it**
  - Get **exclusive** system
  - **Fix clock speed**
  - Control **Affinity / Topology**
    Where does my code/threads/processes run exactly?
  - **Working set size** – code input parameters?!
  - Is result **deterministic**, **reproducible** and **reasonable**
    → Do statistics (Mean, Best, …)

Performance Measurement *Best Practices*

```
 do k = 1 , Nk; do j = 1, Nj
        do i = 1, Ni
              y(i,j,k) = const*
$             ( 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
```

- Postprocessing
  - Documentation
  - Try to understand and explain the result
  - Plan variations to gain more information
  - Many things can be better understood if you plot them (gnuplot, xmgrace)

Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz
Memory Bandwidth 48 GB/s

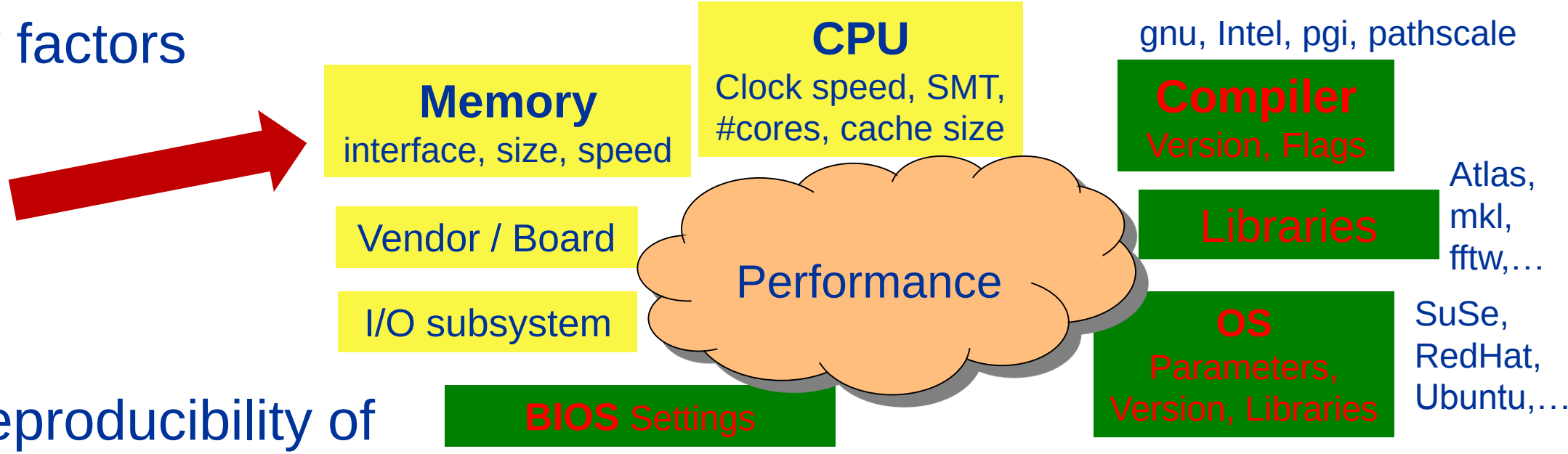


  - Is there a (simple) model which can (qualitatively) explain the performance levels and variations?

What are limiting factors?

Performance
Impact factors

- For a given code/problem performance may be influenced by many factors



- For reproducibility of results all performance critical factors need to be reported!
- Sensibility and stability analysis!
- Statistics - fluctuations between runs

What is hardware performance monitoring?
What is LIKWID?

What is hardware performance monitoring?
What about software performance monitoring?

- People write code and don’t design hardware
- Execution behavior is specified by my code
- Applications are just compiled code. Compilers are always right

not

- *“I know how much data is consumed, it’s my code!”*
- *“I already sum up the number of FLOPs!”*
- *“Of course I overloaded the operators, it’s C++”*

What is hardware performance monitoring?
Overview about HPM

- Performance monitoring units (PMUs) at hardware level
- Introduced for x86 with Intel Pentium (1994)
- Originally used by CPU vendors for hardware validation
- No additional CPU work to process hardware events in PMUs
- Accessing PMUs requires CPU work → Overhead
- Limited number of counters per PMU

Tools for hardware performance monitoring

- Kernel-Interface perf_event
  - Kernel handles management of PMUs (wrong kernel = no support)
  - Counting per OS processes or HW threads
  - Feature-rich interface used by many tools: perf, PAPI, …
- LIKWID
  - Runs in user-space. Uses common kernel interfaces or perf_event
  - Counts per HW thread (knowledge about processes)
  - Derives metrics out of raw counter values

Of course:
Intel VTune

LIKWID

Overview

We got a price of the name ☺

- „Like I Knew What I'm Doing"
- Tool suite for performance-oriented programmers
- Developed by HPC group of RRZE since 2009
- Open source and we are working on it

Goals:

- Provide support for architecture at official release date
- Offer powerful tools for daily work

How to use LIKWID on RWTH systems

- CLAIX18
  - `srun --reservation=aixcel_perf \`
    `  --pty /usr/local_rwth/bin/zsh`
  - `module load likwid`
  - Bash currently not working
- CLAIX16
  - `#BSUB -app likwid`
  - `module load likwid`

LIKWID

System topology

likwid-topology

- System information
- Thread, cache and NUMA topology

```
Socket 0:                                   (Intel i7-4770)
+-----------------------------------------------------------+
| +-------+ +-------+ +-------+ +-------+ |
| |  0 4  | |  1 5  | |  2 6  | |  3 7  | |
| +-------+ +-------+ +-------+ +-------+ |
| +-------+ +-------+ +-------+ +-------+ |
| |  32kB | |  32kB | |  32kB | |  32kB | |
| +-------+ +-------+ +-------+ +-------+ |
| +-------+ +-------+ +-------+ +-------+ |
| | 256kB | | 256kB | | 256kB | | 256kB | |
| +-------+ +-------+ +-------+ +-------+ |
| +-------------------------------------+ |
| |                  8MB                | |
| +-------------------------------------+ |
+-----------------------------------------------------------+
```

Process affinity

likwid-pin -c <cpustr> ./a.out

- Handles common threading solutions
- Simple cpu selection syntax
- `likwid-pin -c S0:0-9@S1:0-9 ./a.out`
  10 processes on socket0 and socket1 (phys. cores first)

LIKWID
HPM with *likwid-perfctr*

- Control application for HPM measurements
- Measurement modes:
  - Start-to-end
  - Timeline (time-based sampling)
  - Stethoscope
  - MarkerAPI (code instrumentation)
- likwid-perfctr –c/–C <cpusel> -g <eventlist> (opts) ./app
  - -c : Measure on cores in cpusel, -C : Measure & pin
  - -g : List of event+counter combis or performance group
- List of all counters and events: likwid-perfctr –e
- Search for some event: likwid-perfctr –E <searchstr>

LIKWID
HPM with *likwid-perfctr*

Profiling:
Start counters
Run application
Stop & Evaluate counters

```
$ likwid-perfctr -C 0,1 -g L2_TRANS_L1D_WB:PMC0 ./app
+-----------------------+---------+--------------+--------------+
|         Event         | Counter |    Core 0    |    Core 1    |
+-----------------------+---------+--------------+--------------+
|  Runtime (RDTSC) [s]  |   TSC   | 2.573182e+00 | 2.573182e+00 |
|    L2_TRANS_L1D_WB    |   PMC0  |    281176518 |    281240170 |
+-----------------------+---------+--------------+--------------+
```

- Event names (in many cases) not intuitive
- Events are architecture-specific
- Some sound promising but return bad counts, others are broken
- More interest in real metrics like volume of loaded data

LIKWID
HPM with *likwid-perfctr*

- LIKWID defines performance groups

  ≈ eventlist + derived metrics + documentation

- List all groups: likwid-perfctr -a

```
$ likwid-perfctr -C 0,1 -g L2 ./app
+-------------------------------+-----------+-----------+
|             Metric            |   Core 0  |   Core 1  |
+-------------------------------+-----------+-----------+
|       Runtime (RDTSC) [s]     |    2.6439 |    2.6439 |
|  L2D write bandwidth [MBytes/s] | 6744.8121 | 6743.6037 |
|  L2D write data volume [GBytes] |   17.8325 |   17.8293 |
|      <there is more>          |           |           |
+-------------------------------+-----------+-----------+
```

Find hot functions (if you don't know)

Only serial analysis

Checks function execution latency

- gprof, Intel compiler, xray (clang) or perf record

```
gcc -pg … && ./a.out && gprof <opts> a.out gmon.out
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 95.70     12.47    12.47                             _work_loop_hsw_corei_fma
  3.45     12.92     0.45                             thread
  0.61     13.00     0.08   990522     0.00     0.00  timestamp
  0.08     13.03     0.01        2     5.00     5.00  sm_work_hsw_corei_fma_2t
  0.00     13.03     0.00      434     0.00     0.00  get_cpu_family
```

And now?

Get a first impression where the time is spent

LIKWID

HPM of functions

Also available for FORTRAN

- Code instrumentation using LIKWID's MarkerAPI

```
#include <likwid.h>
LIKWID_MARKER_INIT; // in serial region
LIKWID_MARKER_REGISTER("Compute"); // in parallel region

LIKWID_MARKER_START("Compute");
<code>
LIKWID_MARKER_STOP("Compute");

LIKWID_MARKER_CLOSE;  // in serial region
```

Recommended: Reduces startup overhead

Multiple regions and nesting allowed

Activate MarkerAPI mode

- Compile with `-DLIKWID_PERFMON`
- `likwid-perfctr -C 0,1 -g L2 -m ./app`

LIKWID

HPM of functions

```
$ likwid-perfctr -C 0,1 -g L2 -m ./app 2000000
Region Compute, Group 1: L2
+-------------------+-----------+-----------+
|    Region Info    |   Core 0  |   Core 1  |
+-------------------+-----------+-----------+
| RDTSC Runtime [s] | 77.724980 | 77.725490 |
|     call count    |      1000 |      1000 |
+-------------------+-----------+-----------+
[…]
+-----------------------------+-------------+-------------+
|             Metric          |   Core 0    |   Core 1    |
+-----------------------------+-------------+-------------+
|       Runtime (RDTSC) [s]   |    77.7250  |    77.7255  |
|     L2 bandwidth [MBytes/s] | 10275.9891  | 10272.4909  | -> SUM: 15977.135
|     L2 data volume [GBytes] |   798.7010  |   798.4344  |
```

If we know that each region executes 2M iterations:

$$\frac{1597.135\ GByte}{1E3 * 2E6\ iterations}$$

$$\approx 798 \frac{Byte}{iterations}$$

LIKWID
HPM + MPI with likwid-mpirun

At CLAIX18 use:

```
--mpi intelmpi
```

- Wrapper for MPI for pinning and HPM

```
$ likwid-mpirun -nperdomain S:1 ./a.out
```

One process per socket on all hosts

```
$ likwid-mpirun -pin S0:0-3_S1:0-3 ./a.out
```

Two processes per node, each using 4 threads

```
$ likwid-mpirun -np <p> -g FLOPS_DP -m ./a.out
```

Measure DP Flops on with all procs (with MarkerAPI)

- Works with Intel MPI, OpenMP, Mvapich and SLURM

```
salloc -N X; likwid-mpirun -np Y ./a.out
```

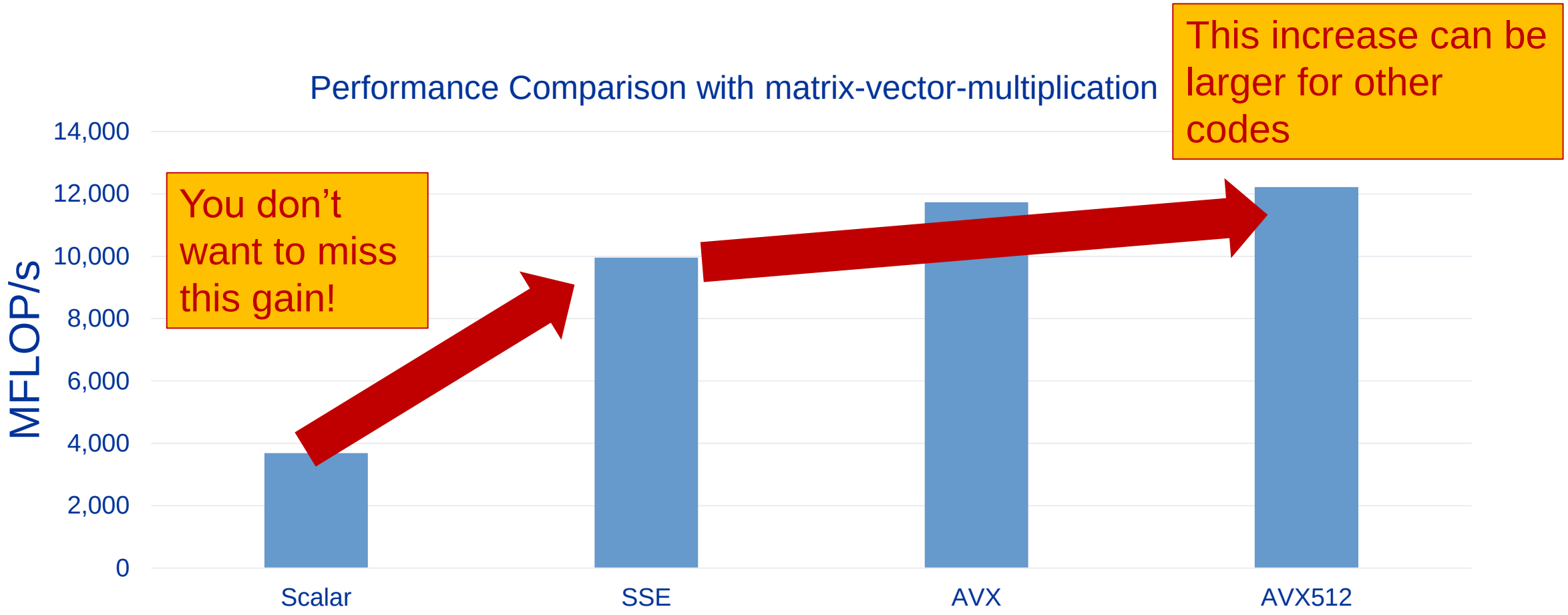
Detection of limiting factors

- Bad vectorization
- Load imbalance
- False sharing in cache hierarchy

Hands on – Bad vectorization

- Modern CPUs gain most of their FP power from vectorization
- Intel compiler does a good job detecting vectorizable code
  - GCC not that good, use –ftree-vectorize –m[avx2|avx512f]
- Mark loops with proper pragmas:
  - #pragma simd
  - #pragma vector align
- Specify build target with -x<target> like -xCORE-AVX512 for Skylake (-xHost sometimes doesn't do the job!)
  (Force AVX512: -qopt-zmm-usage=high)

Hands on – Bad vectorization



Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz

Hands on – Bad vectorization

Almost all FP ops are AVX

- Detect with FLOPS_DP or FLOPS_SP groups

Metric	Core 0	Core 1	Core 2	Core 3
DP MFLOP/s	2204.7856	2214.7243	2213.3518	2209.7715
AVX DP MFLOP/s	2203.4644	2213.3972	2212.0254	2208.4474
Packed MUOPS/s	550.9762	553.4599	553.1168	552.2222
Scalar MUOPS/s	1.1011	1.1060	1.1054	1.1036
Vectorization ratio	99.8006	99.8006	99.8006	99.8006

- Problem: Hardware counts instructions, AVX512 supports masking → Reported numbers might be too high

Detection of limiting factors

- Load imbalance

Dense matrix-vector multiplication in DP
Parallelization

- Let's assume symmetric matrix
  - Avoid unneeded computation by calculating only triangular matrix
  - Distribute columns to threads
  - Problem?
    Can you name it?

```
!$OMP PARALLEL DO PRIVATE(r)
do c = 1 , SIZE
  tmp=x(c)
  do r = 1 , c
     y(r)=y(r) + A(r,c)* tmp
  enddo
enddo
!$OMP END PARALLEL DO
```

Dense matrix-vector multiplication in DP

Parallelization

```
$ likwid-perfctr -C 0,1,2 -g L2 -m ./a.out
--------------------------------------------------------------------------------
CPU type:       Intel Core SandyBridge EN/EP processor
CPU clock:      3.09 GHz
--------------------------------------------------------------------------------
========================
Group 1: Region Compute
========================
+-----------------------+---------+--------------+--------------+--------------+
|         Event         | Counter |    Core 0    |    Core 1    |    Core 2    |
+-----------------------+---------+--------------+--------------+--------------+
|   INSTR_RETIRED_ANY   |  FIXC0  | 2.626800e+08 | 3.187585e+08 | 3.780255e+08 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  | 4.972802e+08 | 4.961411e+08 | [illegible]  |
|  CPU_CLK_UNHALTED_REF |  FIXC2  | 4.972801e+08 | 4.961404e+08 | 4.933714e+08 |
|    L1D_REPLACEMENT    |   PMC0  | 5.490278e+07 | 3.927353e+07 | 2.364295e+07 |
|      L1D_M_EVICT      |   PMC1  | 2.920200e+04 | 2.876600e+04 | 2.861000e+04 |
+-----------------------+---------+--------------+--------------+--------------+
```

Always use a reasonable metric for “work”!

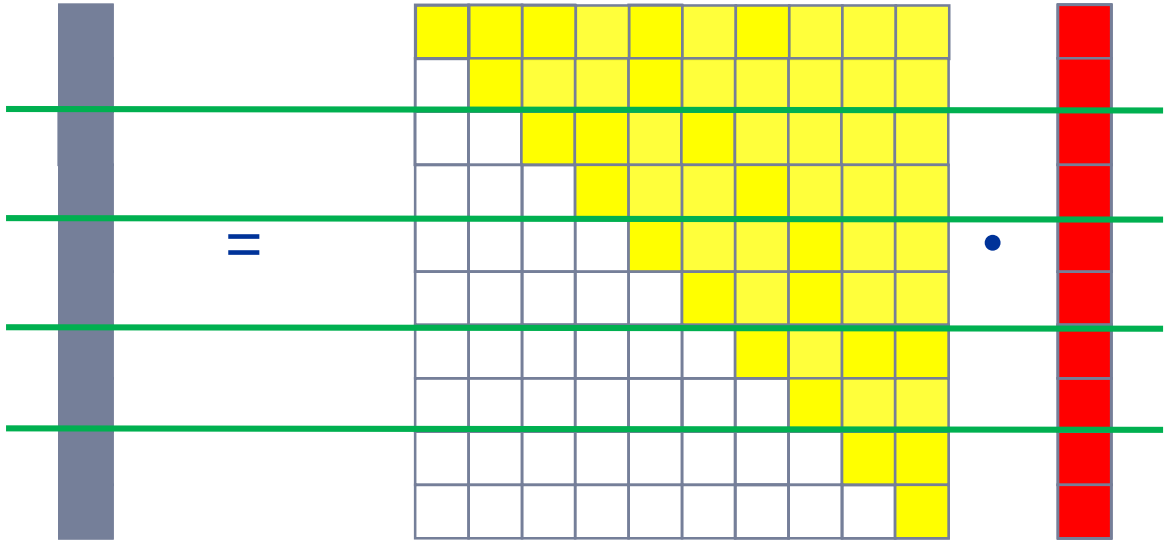
Number of instructions raise with CPU number

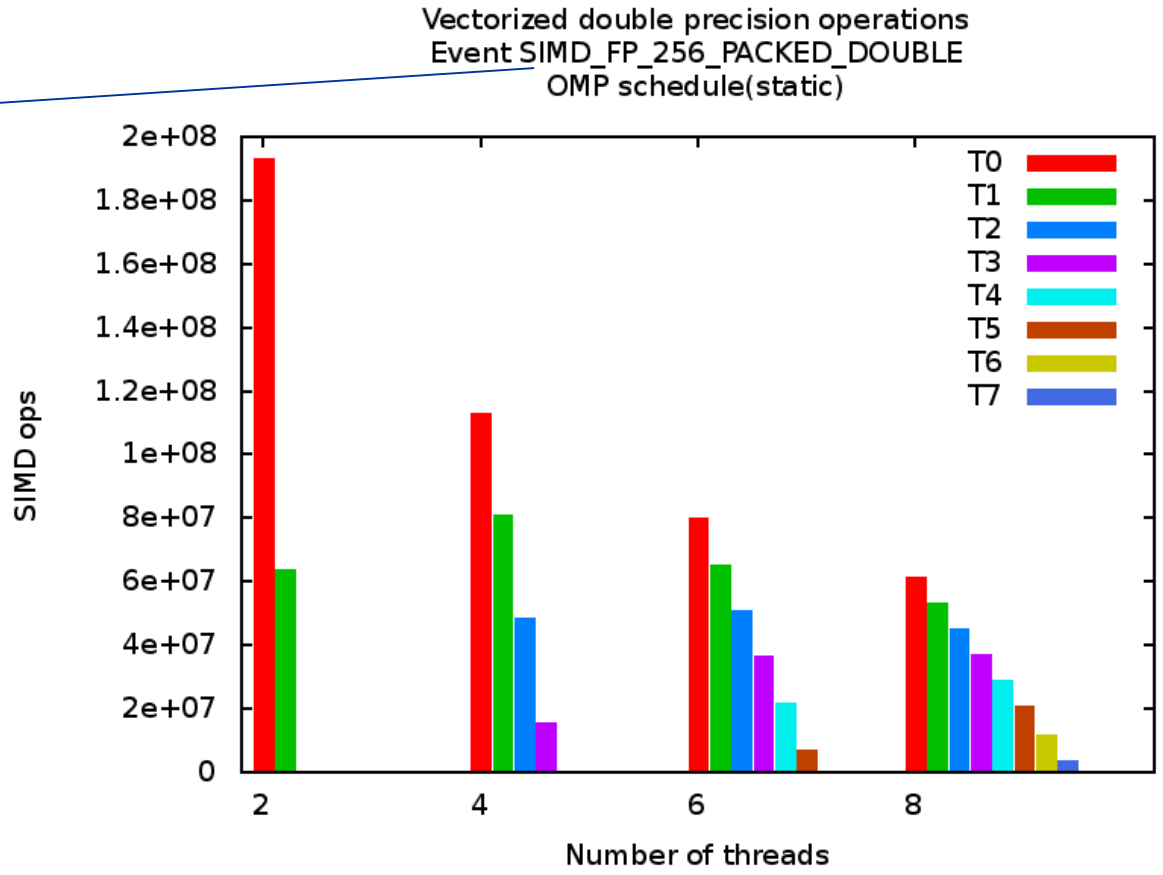
First CPU loads most data into L1

Same amount of CLs evicted from L1

Dense matrix-vector multiplication in DP
Parallelization

- Here we look at 256 bit wide vector operations → AVX
- First thread "works" most





Intel SandyBridge

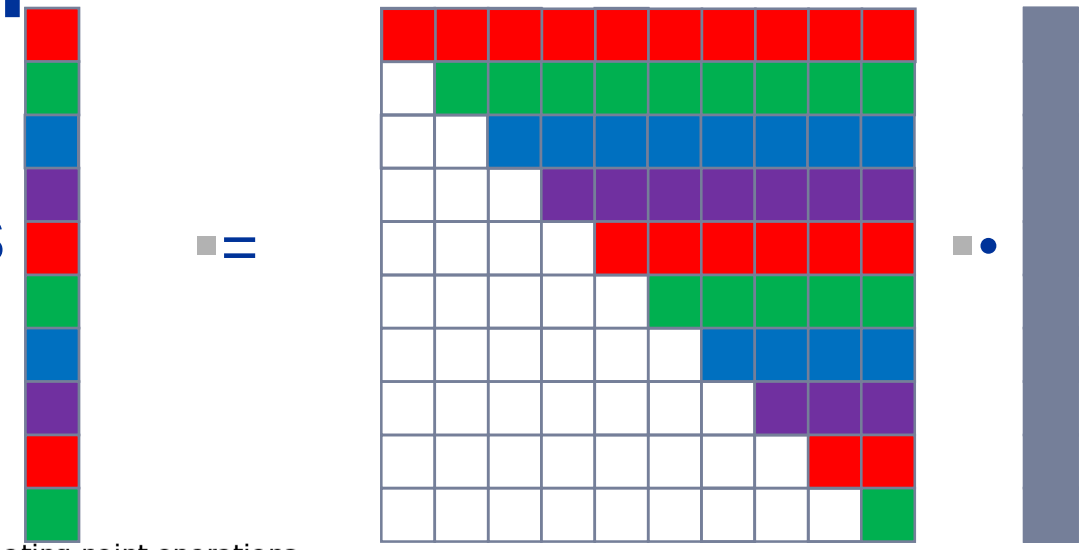
Dense matrix-vector multiplication in DP
Parallelization

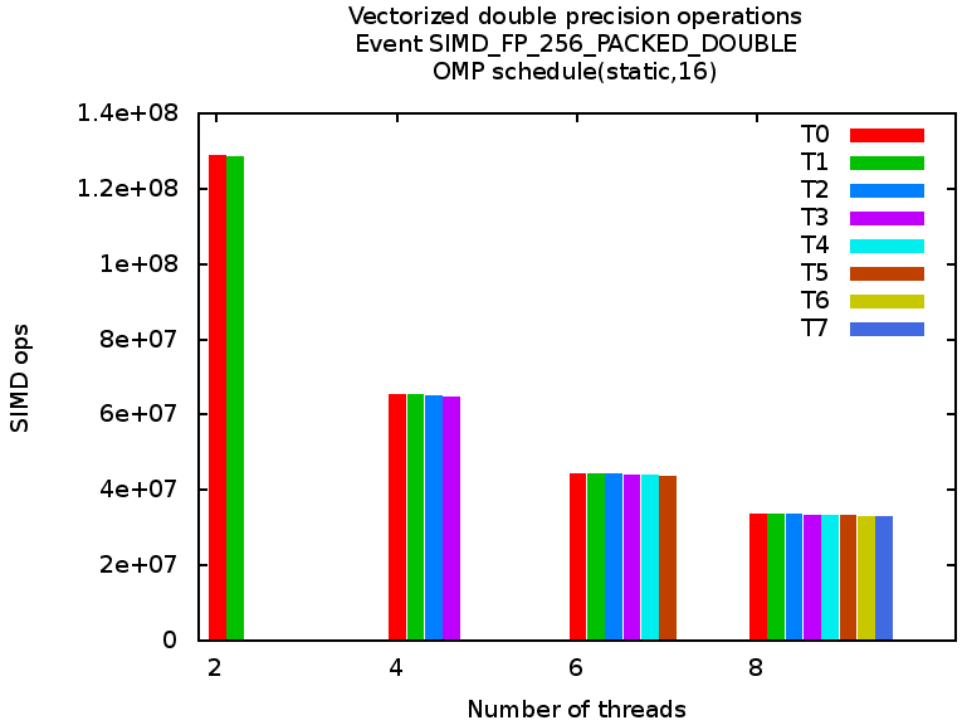
- How to balance the workload for every thread?
- Over-engineering:
  - Different data layout
  - Use smaller chunks of work (split rows/columns in multiple chunks?)
- Simple: Change OpenMP schedule `schedule(kind, chunksize)`
  - dynamic: actively balance chunks of work → Overhead
  - static: pre-compute chunks of work → No Overhead
  - chunksize parameter describes size of chunks

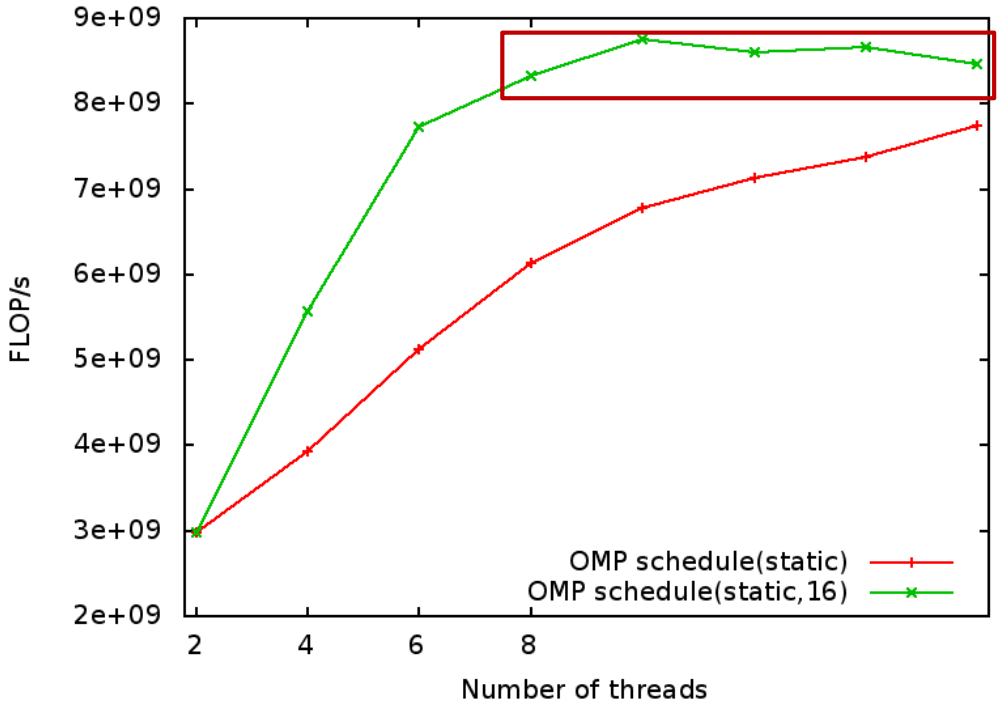
Dense matrix-vector multiplication in DP

Parallelization

- Reduce chunk of work to a few rows
- Not optimal, but sufficient!





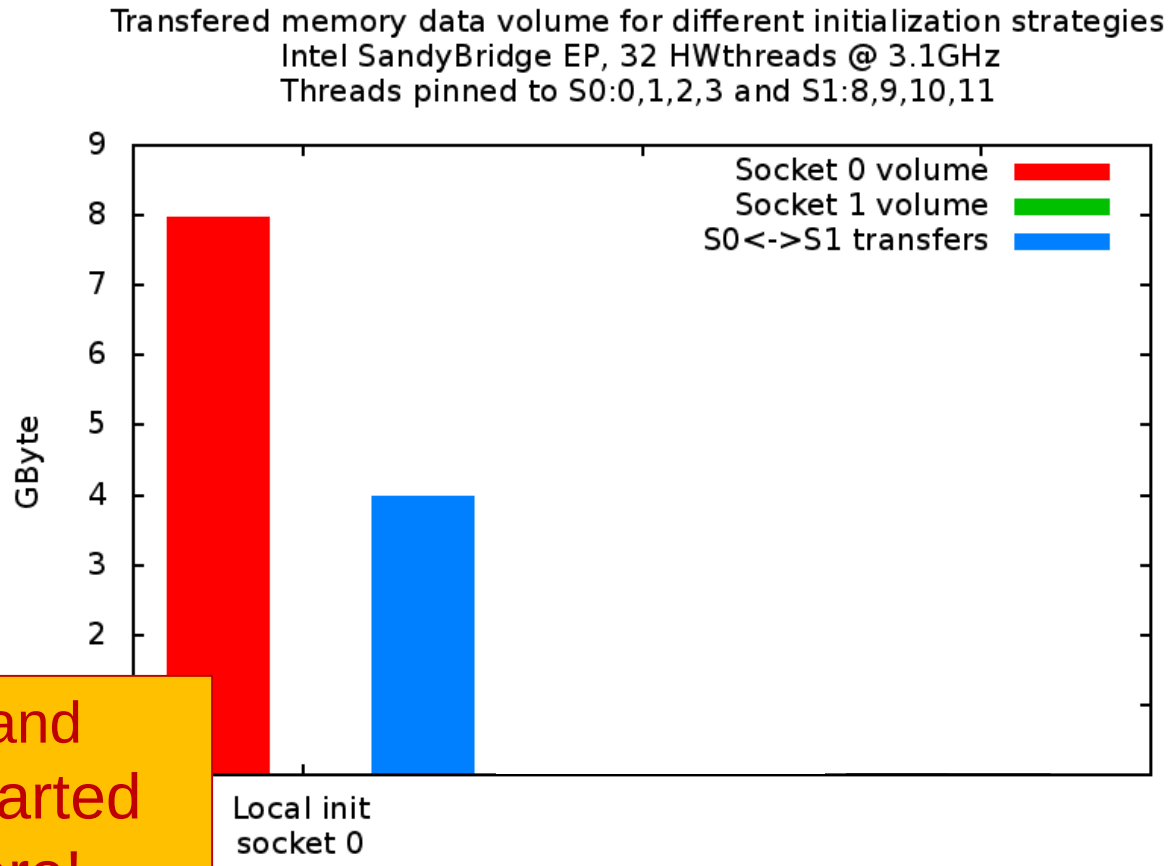


Saturate memory bandwidth

Intel SandyBridge

Dense matrix-vector multiplication in DP
Parallelization

- malloc() does not locate data
- First access defines location (first touch)
- Data locality crucial for high performance on ccNUMA systems



Not always, but true for GCC and Intel/LLVM OpenMP: Once started and pinned threads stay there!

```
numactl -i <nodesel> ./a.out
```

- Simplification: Use same work sharing for allocation and work!

Detection of limiting factors

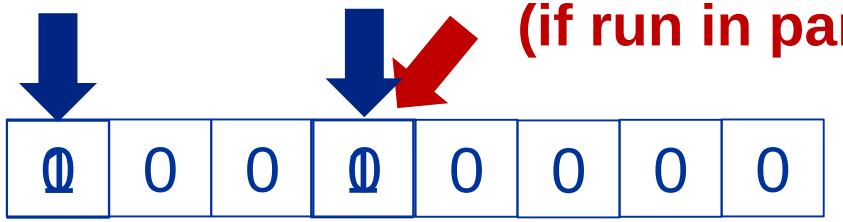
- False-sharing

Hands on – False-sharing

- **Serial histogram code**

```
int hist[8] = { 0 };
for(j=0; j<1000000000; ++j) {
    hist[rand_r(&seed) &
0x7]++;
}
```

Possible data race!
(if run in parallel)



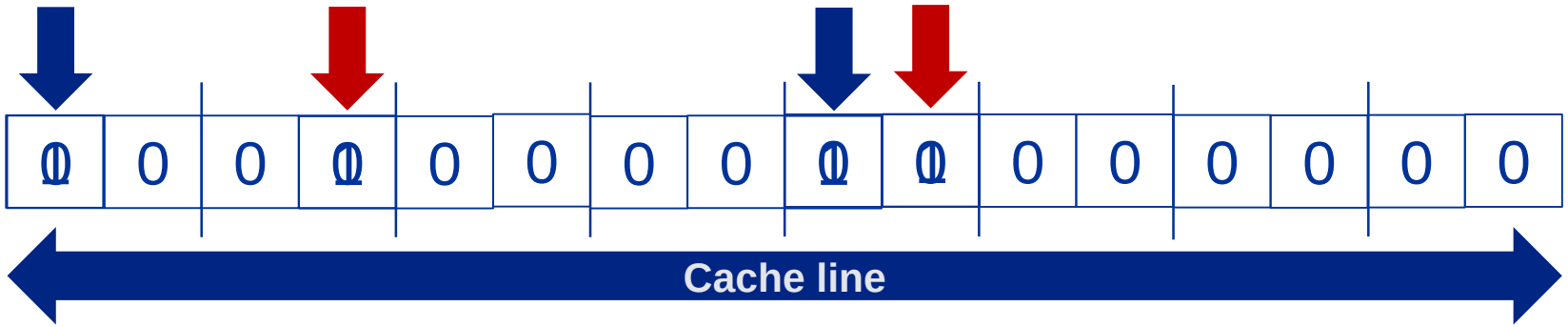
Hands on – False-sharing

Parallelize histogram code

- **Solution: Own histogram per thread**

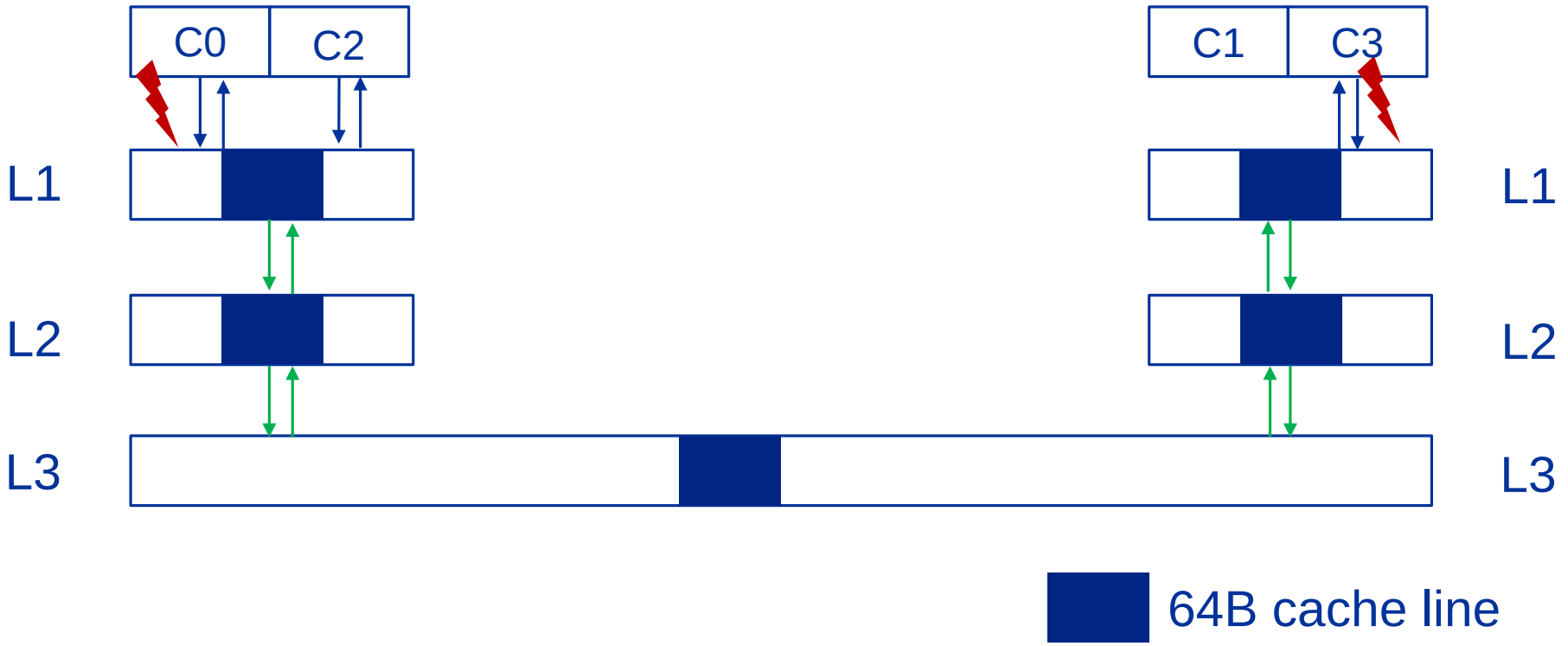
```
int hist[8][MAX_THREADS] = { 0 };
#pragma omp parallel for firstprivate(seed)
for(j=0; j<1000000000; ++j)
    hist[rand_r(&seed) & 0x7][tid]++;
// combine thread histograms
```

No data race! But….



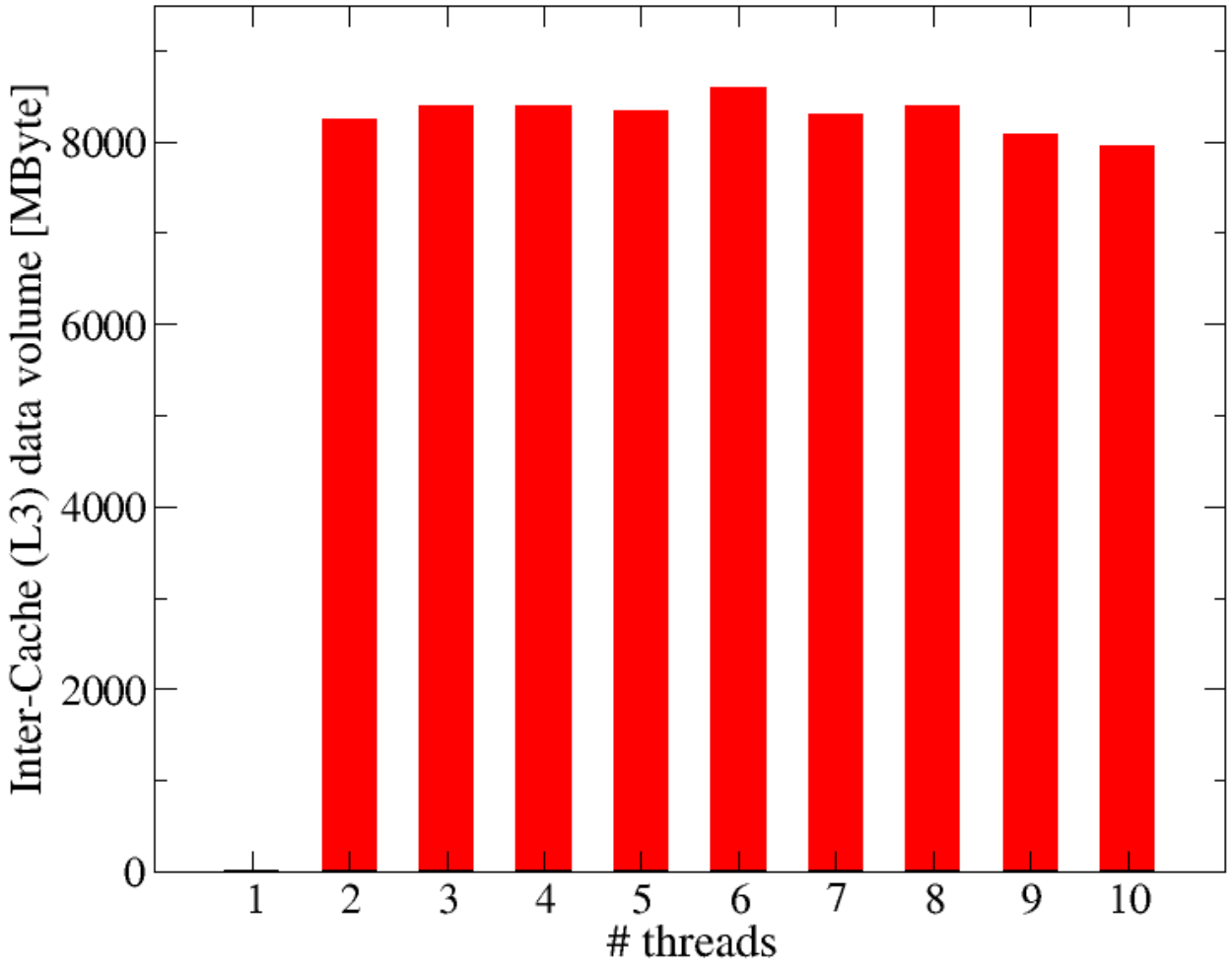
Hands on – False-sharing

Parallelize histogram code



Hands on – False-sharing

Parallelize histogram code



Single thread:

4.85 s

10 threads:

7.58 s

After Optimization with thread-local histogram

10 threads: 0.61 s

Intel IvyBridge EP

Hands on – False-sharing

Problems

- CL sharing is normal in almost every code
  - Global data structures (stop criterion, memory adresses, …)
  - Common structs (static information)
- Cannot differentiate between *required* shared CL and *falsly* shared CL
- Counters not accurate (Haswell: up to 50% deviation)
- No information about cache flushes / memory barriers

Useful performance groups

Topic	Perf. group
FP operations	FLOPS_DP FLOPS_SP
Memory traffic	MEM
Quality of cache usage	L2CACHE L3CACHE
Energy usage	ENERGY
TLBs	TLB_DATA TLB_INSTR
Branch prediction	BRANCH
In-core stalls	CYCLE_ACTIVITY

Thank you for your attention!

Regionales RechenZentrum Erlangen [RRZE]

Martensstraße 1, 91058 Erlangen

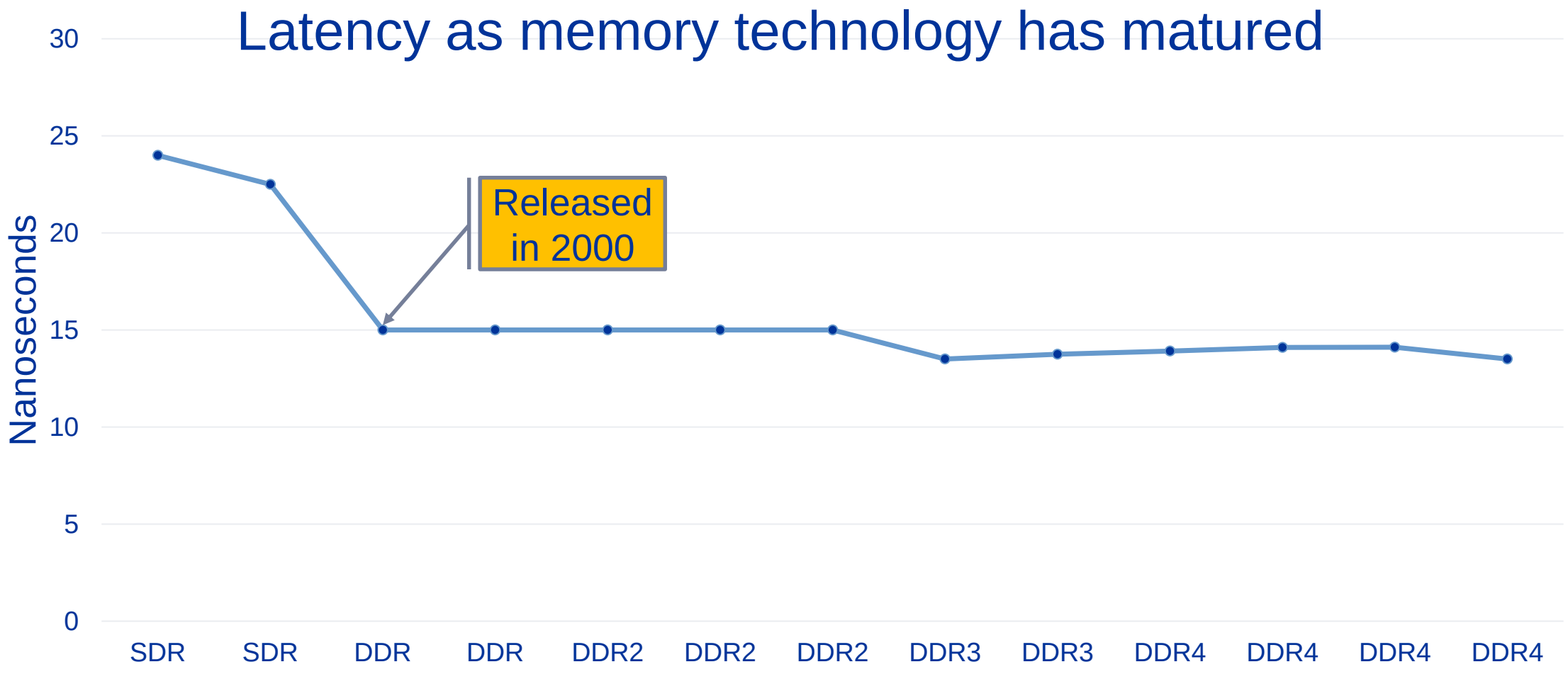
http://www.rrze.fau.de

https://github.com/RRZE-HPC/likwid/wiki

Detection of limiting factors

- Bandwidth limitation

Motivation
A Short look back

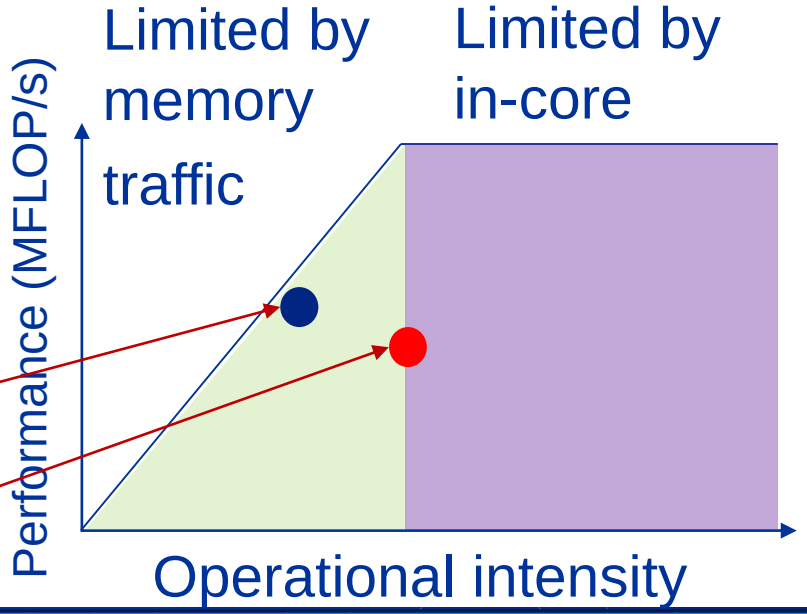


Hands on – Bandwidth limitations

- Main limitation nowadays is memory bandwidth/latency
- How to detect code is limited by memory bandwidth?
  - You need to know how much data you need to load/store from/to memory
  - Use a model like Roofline or ECM (more insight but more work)
  - Data for Roofline: MEM_DP, MEM_SP

Clearly memory bound

And now? Not clear at all



Jacobi 2D 5pt stencil (DP)

Data traffic analysis

sweep

```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (  x(j-1,k) + x(j+1,k) &
                     +  x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

Available in cache (used 2 iterations before)

LD x(j+1,k)

LD x(j,k+1)

LD+ST y(j,k)
(incl. write allocate)

LD x(j,k-1)

Naive balance (incl. write allocate):

`x( :, :)` : 3 LD +
`y( :, :)` : 1 ST+ 1LD

→ B_C = 5 Words / LUP → B_C = 40 B / LUP
(assuming double precision)

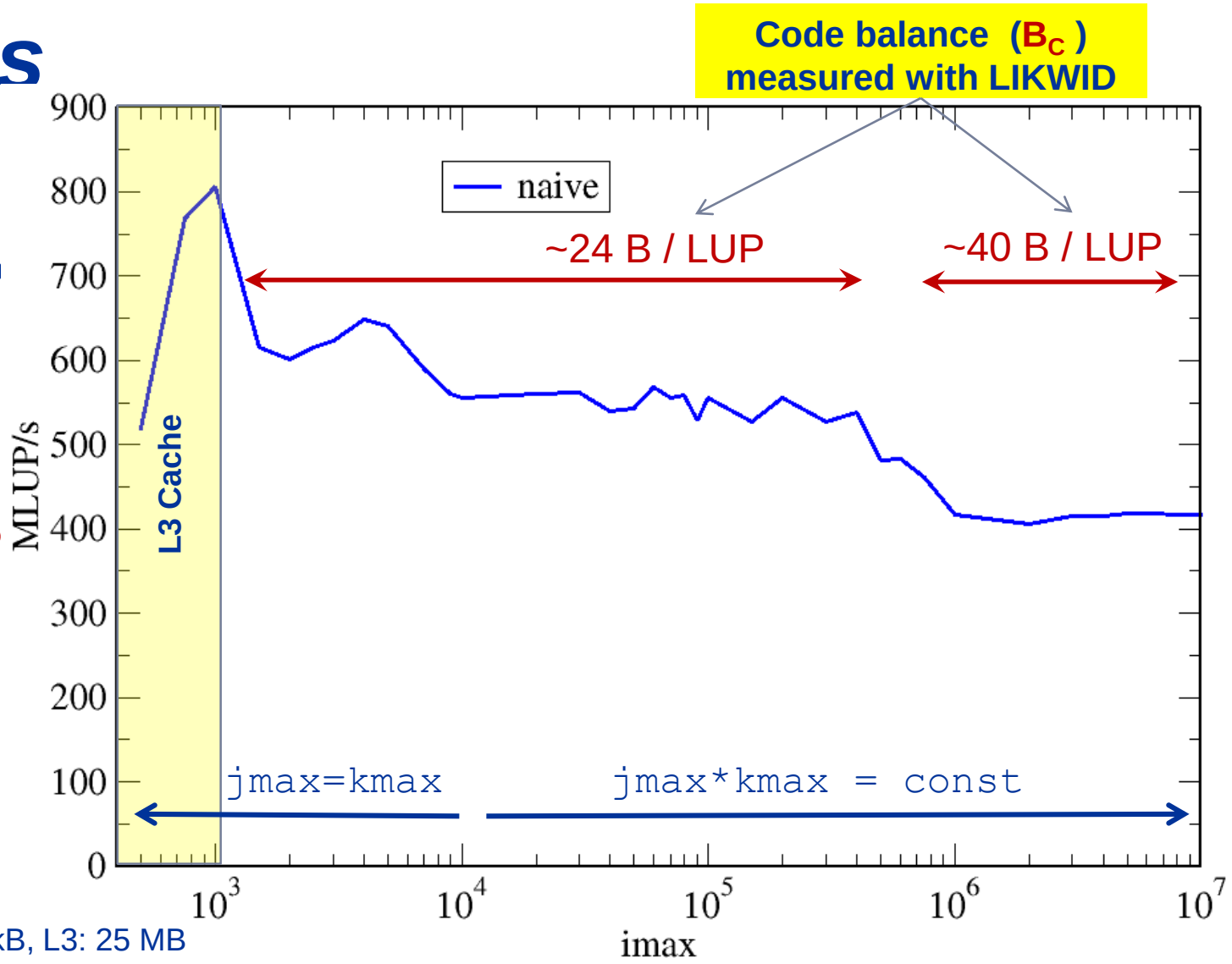
Reminder "write allocate":

If a cache line is not present in L1 cache at a store, the CL needs to loaded first into L1.

Jacobi 2D 5pt stencil (DP)
Data traffic analysis

Questions:

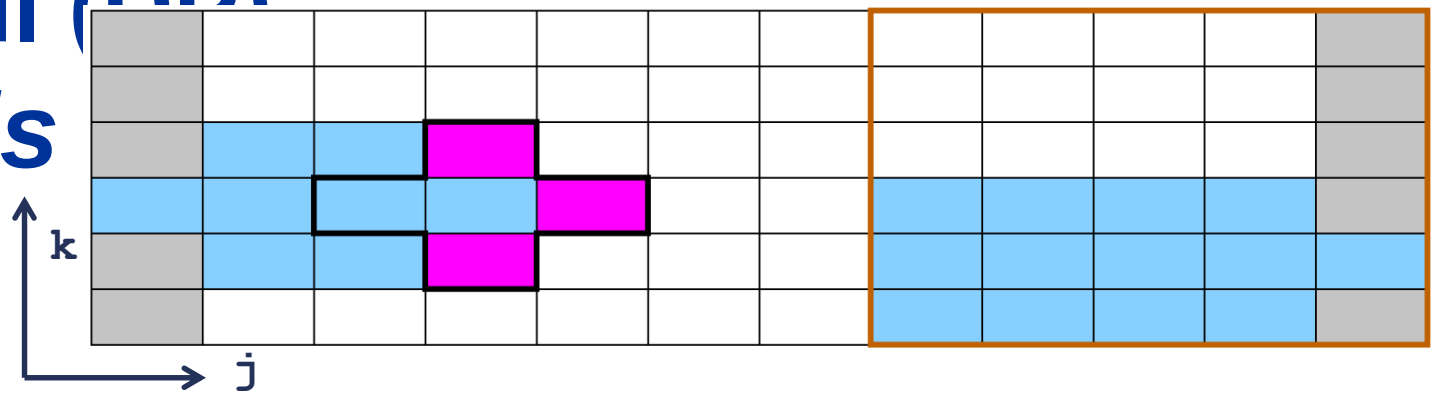
1. How to achieve 24 B/LUP
2. How to sustain > 600 MLUP/s for `jmax>10^4` ?
3. Why 24 B/LUP anyway?? Naïve balance was 40B/LUP!



Intel Xeon E5-2690 v2 @ 3 GHz, ifort V13.1, L1: 32kB, L2: 256kB, L3: 25 MB

Jacobi 2D 5pt stencil (DP)
Data traffic analysis

Reduce inner (j-) loop dimension successively



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

Jacobi 2D 5pt stencil (DP)
Data traffic analysis

y: (1 LD + 1 ST) / LUP

x: 1 LD / LUP

```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * ( x(j-1,k) + x(j+1,k) &
                     +   x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

B_C = 24 B / LUP

YES

3 * jmax * 8B < CacheSize/2

"Layer condition" fulfilled?

NO

y: (1 LD + 1 ST) / LUP

```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * ( x(j-1,k) + x(j+1,k) &
                     +   x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

x: 3 LD / LUP

B_C = 40 B / LUP

Jacobi 2D 5pt stencil (DP)
Establish Layer condition for all `jmax`

- Apply spatial blocking of j-loop

Take care, loop over the blocks is most outer loop now

```
do jb=1,jmax,jblock !   Assume jmax is multiple of jblock
  do k=1,kmax
    do j= jb, (jb+jblock-1) ! Length of inner loop: jblock
      y(j,k) = const * (x(j-1,k) + x(j+1,k) &
                       +  x(j,k-1) + x(j,k+1) )
    enddo
  enddo
enddo
```

- Determine for given `CacheSize` an appropriate `jblock`:

New layer condition (blocking)

`3 * jblock * 8B < CacheSize/2`

→

`jblock < CacheSize / 48 B`

Jacobi 5pt stencil (DP)
Blocking for which cache?

Intel Xeon E5-2690 v2 @ 3 GHz, ifort V13.1,
L1: 32kB, L2: 256kB, L3: 25 MB

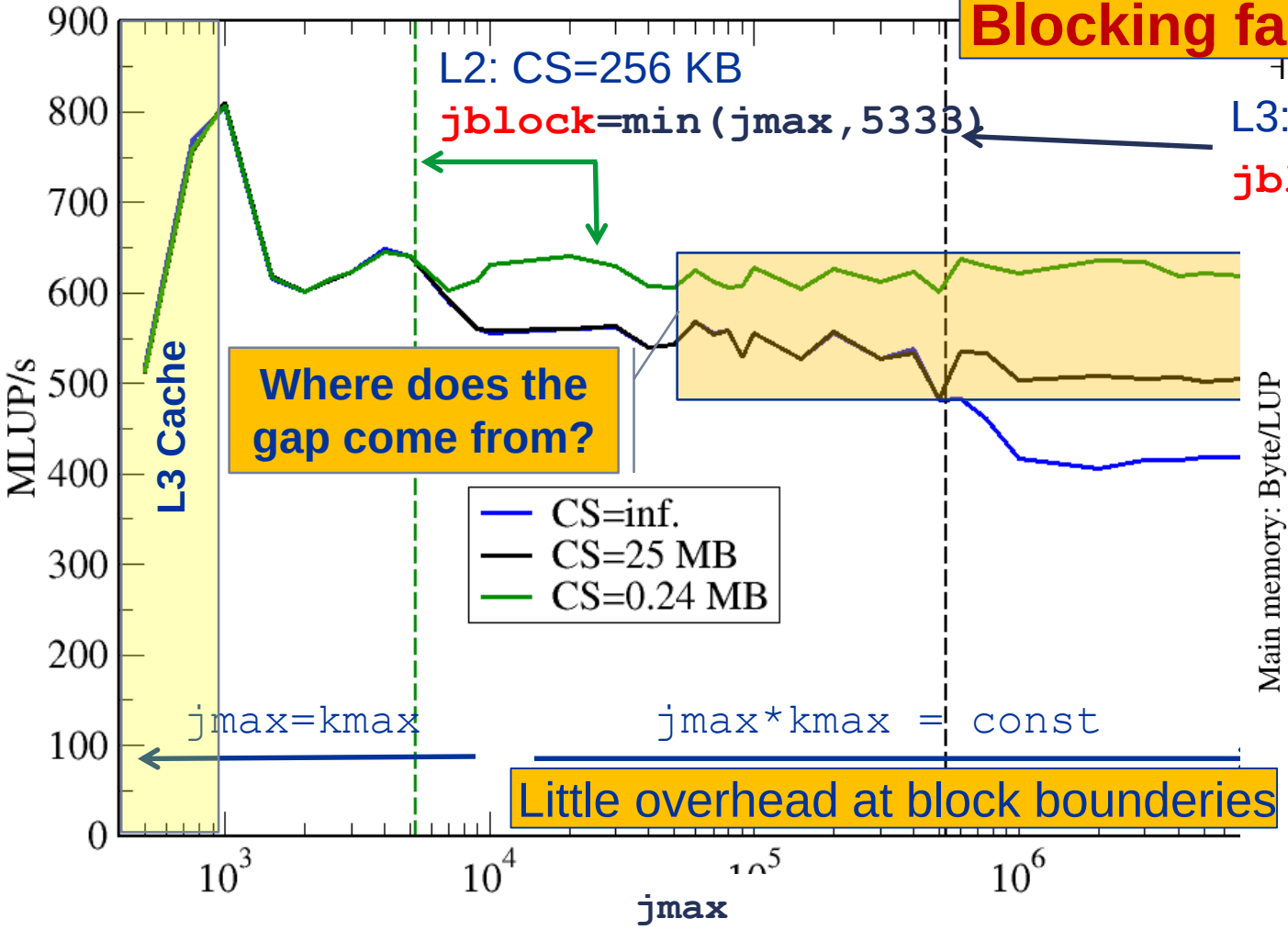
Blocking factor (CS=25MB) too large!

L2: CS=256 KB

`jblock=min(jmax,5333)`

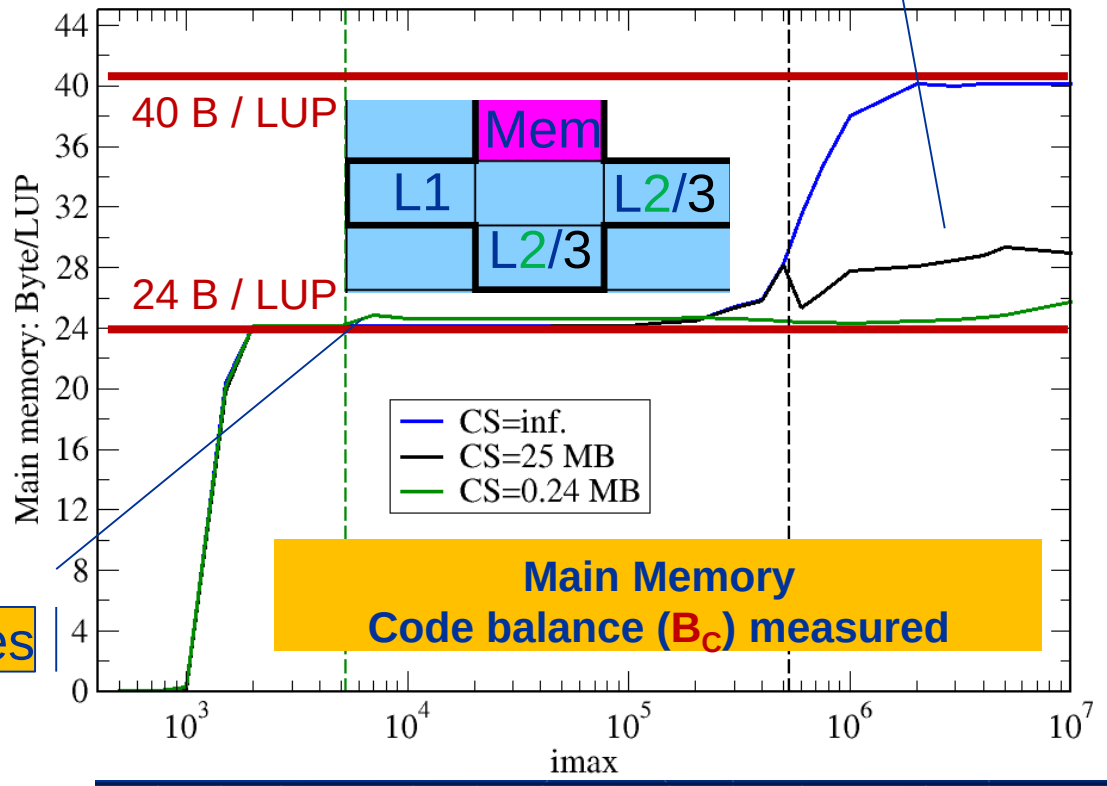
L3: CS=25 MB

`jblock=min(jmax,533333)`



Where does the gap come from?

Little overhead at block bounderies



Main Memory
Code balance (B_C) measured

Jacobi 2D 5pt stencil (DP)
What about 3D 7pt stencil?

3*jmax * 8B < CacheSize/2

```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (x(j-1,k) + x(j+1,k) &
                     +  x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

B_C = 24 B / LUP

"Layer condition" OK →
5 accesses to `x()` served by cache

```
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = const * (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
```

B_C = 24 B / LUP

Same for version with spatial blocking

3*jmax *imax * 8B < CacheSize/2

Jacobi 3D 7pt stencil (DP)
Blocking, layer condition and threading?

```
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
        y(i,j,k) = const *(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

enddo
  !$OMP END PARALLEL DO
```

"Layer condition" (j-Blocking))

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

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

Intel Xeon E5-2690 v2 @ 3 GHz, ifort V13.1,
L1: 32kB, L2: 256kB, L3: 25 MB

Maximum memory bandwidth $b_S = 48\ \mathrm{GB/s}$

Best performance:

$$P = \frac{b_S}{B_C} = 2000\ \mathrm{MLUP/s}$$