NHR FAU

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

LIKWID Performance Tools

EXA2PRO-EoCoE joint workshop

Thomas Gruber
NHR@FAU, RRZE, Erlangen

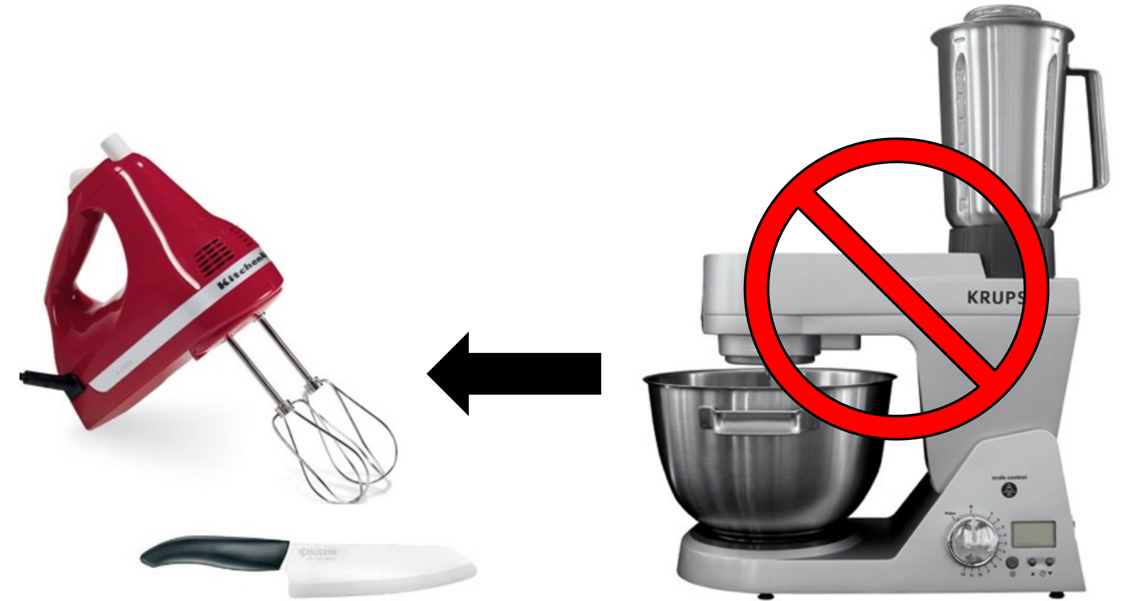
Tools for the daily life

- Read the hardware topology
- Flexible and comprehensive pinning functionality
- Easy-to-use hardware performance event measurements

- Runtime system cleanup
- System adaption
- Micro-benchmarking

What is LIKWID?

- A toolset for performance-oriented developers/users



- Read system topology
- Place threads according system topology (affinity domains)
- Run micro-benchmarks to check system features
- Measure hardware events during application runs
- Determine energy consumption
- Manipulate CPU/Uncore frequencies

LIKWID Tools overview

- likwid-topology – Read topology of current system
- likwid-pin – Pin threads to hardware threads
- likwid-perfctr – Hardware performance monitoring (HPM) tool
- likwid-powermeter – Measure energy consumption
- likwid-memsweeper – Clean up filesystem cache and LLC
- likwid-setFrequencies – Manipulate CPU/Uncore frequency
- likwid-features – Manipulate hardware settings (prefetchers)
- likwid-bench – Microkernel benchmark tool

likwid-topology

```
$ likwid-topology
--------------------------------------------------------------------------------
CPU name:   Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
CPU type:   Intel Xeon Broadwell EN/EP/EX processor
CPU stepping:  1
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:             2
Cores per socket:   10
Threads per core:   1
--------------------------------------------------------------------------------
HWThread    Thread      Core        Socket      Available
0           0           0           0           *
1           0           1           0           *
[…]
--------------------------------------------------------------------------------
Socket 0:       ( 0 1 2 3 4 5 6 7 8 9 )
Socket 1:       ( 10 11 12 13 14 15 16 17 18 19 )
```

likwid-topology

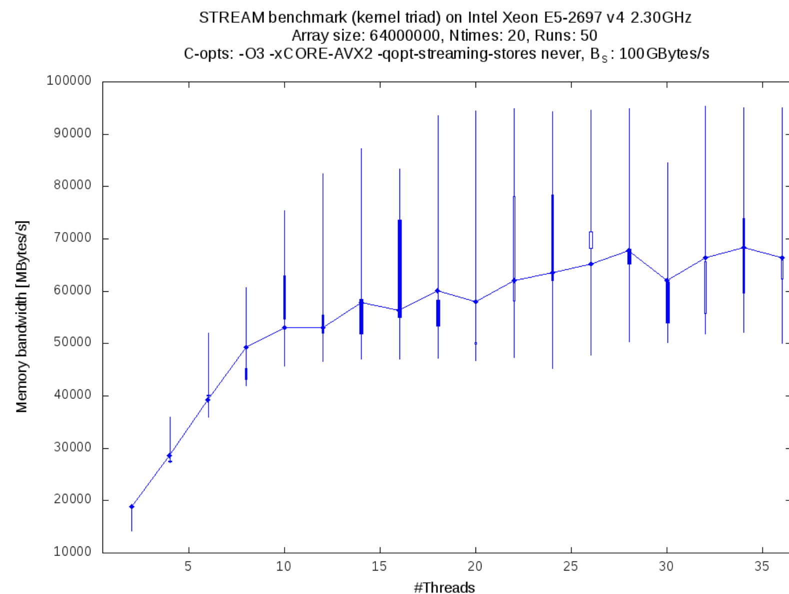
```
********************************************************************************
Cache Topology
********************************************************************************
Level:          1
Size:           32 kB
Cache groups:   ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 )
( 12 ) ( 13 ) ( 14 ) ( 15 ) ( 16 ) ( 17 ) ( 18 ) ( 19 )
--------------------------------------------------------------------------------
Level:          2
Size:           256 kB
Cache groups:   ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 )
( 12 ) ( 13 ) ( 14 ) ( 15 ) ( 16 ) ( 17 ) ( 18 ) ( 19 )
--------------------------------------------------------------------------------
Level:          3
Size:           25 MB
Cache groups:   ( 0 1 2 3 4 5 6 7 8 9 ) ( 10 11 12 13 14 15 16 17 18 19 )
--------------------------------------------------------------------------------
```

likwid-topology

```
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:     2
--------------------------------------------------------------------------------
Domain:           0
Processors:       ( 0 1 2 3 4 5 6 7 8 9 )
Distances:        10 21
Free memory:      29374.3 MB
Total memory:     32079.4 MB
--------------------------------------------------------------------------------
Domain:           1
Processors:       ( 10 11 12 13 14 15 16 17 18 19 )
Distances:        21 10
Free memory:      30176.2 MB
Total memory:     32253.8 MB
--------------------------------------------------------------------------------
```

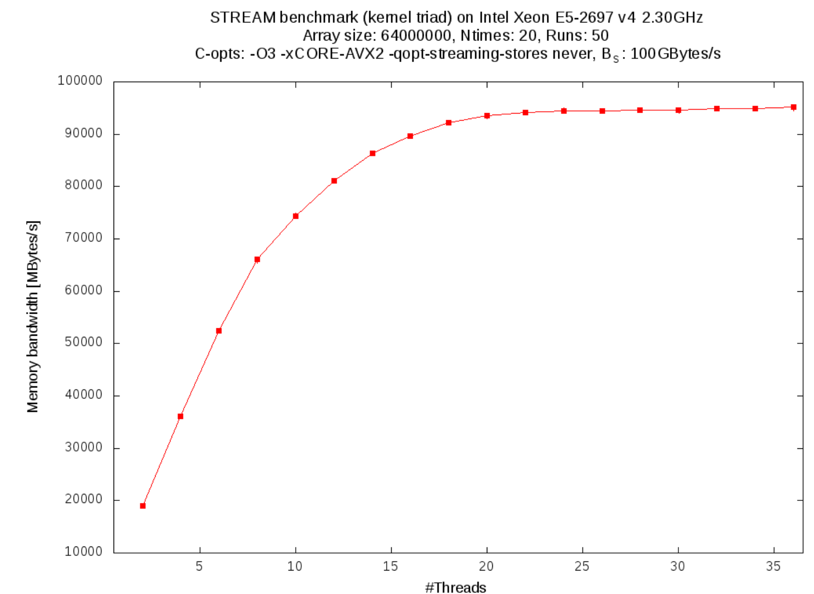
likwid-pin

- OS task scheduler places tasks (=processes/threads) on HW threads
- OS scheduler moves tasks to different HW threads from time to time
  - Cache trashing
  - Changed inter-thread communication latency
  - Overloading and overheating of single HW threads



Affinity control

- Reduce set of possible HW threads
- Force tasks to stay on distinct HW threads



likwid-pin

- Many tools exist for limiting the set of HW threads (`taskset`, `numactl`, ...)
- `likwid-pin` = `taskset` + „forcing tasks on distinct HW threads“

- `$ likwid-pin -c 0-2 ./a.out`
  - Limit set of possible HW threads to 0-2
  - As soon as a thread is started by `a.out`:
    - Reduce set of possible HW threads for master to 0
    - Force thread on next HW thread in initial set → 1
    - Next thread is placed on last HW thread → 2
    - More threads? Round-robin pinning in initial set
- Works for most threading solutions: Pthreads, OpenMP, …

What about more complicated cases? Like one thread per socket?

likwid-pin

```
$ likwid-pin -p
Domain N:  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
Domain S0: 0,1,2,3,4,5,6,7,8,9
Domain S1: 10,11,12,13,14,15,16,17,18,19
Domain C0: 0,1,2,3,4,5,6,7,8,9
Domain C1: 10,11,12,13,14,15,16,17,18,19
Domain M0: 0,1,2,3,4,5,6,7,8,9
Domain M1: 10,11,12,13,14,15,16,17,18,19
```

N: node
S: socket
C: last level cache
M: NUMA domain

```
$ likwid-pin -c M1:0-3 -p
10,11,12,13

$ likwid-pin -c M0:0-3@M1:0-3 -p
0,1,2,3,10,11,12,13
```

```
$ likwid-pin -c M1:0-1 ./a.out
[pthread wrapper]
[pthread wrapper] MAIN -> 0
[pthread wrapper] PIN_MASK: 0->1
[pthread wrapper] SKIP MASK: 0x1
       threadid 47168122029824 -> SKIP
Start OpenMP threads
       threadid 47168145868672 -> core 11 - OK
Rank 0 Thread 0 running on Node m1010 core 10
Rank 0 Thread 1 running on Node m1010 core 11
```

Hardware performance monitoring (likwid-perfctr)

- Increasing complexity of applications
- Multiple abstraction layers between apps and the OS
- Hardware counters run side-by-side with the application → no overhead
- Configurable events at different units (set of counters)
- Different 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 these HW threads
  - `-C`: measure on & pin to these HW threads
  - `-g`: event set of performance group (event set + derived metrics)

likwid-perfctr

```
$ likwid-perfctr -C M0:0-1 -g FLOPS_DP ./a.out
--------------------------------------------------------------------------------
CPU name:  Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
CPU type:  Intel Xeon Broadwell EN/EP/EX processor
CPU clock: 2.19 GHz
--------------------------------------------------------------------------------
<application output>
--------------------------------------------------------------------------------
Group 1: FLOPS_DP
+------------------------------------------+---------+-------------+------------+
|                  Event                   | Counter |   Core 0    |   Core 1   |
+------------------------------------------+---------+-------------+------------+
|            INSTR_RETIRED_ANY             |  FIXC0  |  7316892954 | 2381371909 |
|          CPU_CLK_UNHALTED_CORE           |  FIXC1  | 10767241400 | 4119973243 |
|          CPU_CLK_UNHALTED_REF            |  FIXC2  |  7775413294 | 3164480264 |
| FP_ARITH_INST_RETIRED_128B_PACKED_DOUBLE |   PMC0  |           0 |          0 |
|    FP_ARITH_INST_RETIRED_SCALAR_DOUBLE   |   PMC1  |  2500000019 | 100000023  |
| FP_ARITH_INST_RETIRED_256B_PACKED_DOUBLE |   PMC2  |           0 |          0 |
+------------------------------------------+---------+-------------+------------+
```

Raw events are not intuitively named.

Raw counts might not be comparable

likwid-perfctr

```
$ likwid-perfctr -C M0:0-1 -g FLOPS_DP ./a.out (cont.)
+----------------------+-----------+-----------+
|        Metric        |  Core 0   |  Core 1   |
+----------------------+-----------+-----------+
|  Runtime (RDTSC) [s] |    6.9541 |    6.9541 |
| Runtime unhalted [s] |    7.7226 |    3.9848 |
|      Clock [MHz]     | 3052.0039 | 3044.3406 |
|          CPI         |    1.2251 |    1.6261 |
|     DP [MFLOP/s]     |  790.8997 |  287.5999 |
|   AVX DP [MFLOP/s]   |         0 |         0 |
|   Packed [MUOPS/s]   |         0 |         0 |
|   Scalar [MUOPS/s]   |  790.8997 |  287.5999 |
|  Vectorization ratio |         0 |         0 |
+----------------------+-----------+-----------+
```

Users are commonly interested in derived metrics

Metrics can be compared and referred back to the application e.g. data volume / iterations

But what if we are interested in a specific loop nest, function call, … ?

MarkerAPI (Code instrumentation)

- LIKWID provides an API to restrict measurements to specific code regions

```
#include <likwid-marker.h>
LIKWID_MARKER_INIT; // in serial region
LIKWID_MARKER_REGISTER("bench"); // in parallel region
LIKWID_MARKER_START("bench");
<code>
LIKWID_MARKER_STOP("bench");
LIKWID_MARKER_CLOSE;  // in serial region
```

- Compile with `-I${LIKWID_INCDIR}`
  `-L${LIKWID_LIBDIR}`
  `-DLIKWID_PERFMON`
- And link with `-llikwid`

MarkerAPI available for:
- Fortran90
- Java
- Python
- Lua
- Julia

likwid-perfctr (MarkerAPI)

```
$ likwid-perfctr -C M0:0-1 -g FLOPS_DP -m ./a.out
Region bench, Group 1: FLOPS_DP
+-------------------+----------+----------+
|    Region Info    |  Core 0  |  Core 1  |
+-------------------+----------+----------+
| RDTSC Runtime [s] | 1.435659 | 1.435709 |
|     call count    |        1 |        1 |
+-------------------+----------+----------+
+------------------------------------------+---------+------------+------------+
|                   Event                  | Counter |   Core 0   |   Core 1   |
+------------------------------------------+---------+------------+------------+
|             INSTR_RETIRED_ANY            |  FIXC0  | 2377031000 | 2375010000 |
|           CPU_CLK_UNHALTED_CORE          |  FIXC1  | 4315301000 | 4312481000 |
|           CPU_CLK_UNHALTED_REF           |  FIXC2  | 3091968000 | 3089982000 |
| FP_ARITH_INST_RETIRED_128B_PACKED_DOUBLE |   PMC0  |          0 |          0 |
|    FP_ARITH_INST_RETIRED_SCALAR_DOUBLE   |   PMC1  | 1000000000 | 1000000000 |
| FP_ARITH_INST_RETIRED_256B_PACKED_DOUBLE |   PMC2  |          0 |          0 |
+------------------------------------------+---------+------------+------------+
```

likwid-perfctr (MarkerAPI)

```
$ likwid-perfctr -C M0:0-1 -g FLOPS_DP -m ./a.out (cont.)
+----------------------+-----------+-----------+
|        Metric        |   Core 0  |   Core 1  |
+----------------------+-----------+-----------+
|  Runtime (RDTSC) [s] |    1.4357 |    1.4357 |
| Runtime unhalted [s] |    1.9660 |    1.9648 |
|      Clock [MHz]     | 3063.3425 | 3063.3082 |
|          CPI         |    1.8154 |    1.8158 |
|     DP [MFLOP/s]     |  696.5442 |  696.5200 |
|   AVX DP [MFLOP/s]   |         0 |         0 |
|   Packed [MUOPS/s]   |         0 |         0 |
|   Scalar [MUOPS/s]   |  696.5442 |  696.5200 |
|  Vectorization ratio |         0 |         0 |
+----------------------+-----------+-----------+
```

Where to start?

- Run application for first impression (`L2,L3,MEM,DATA,FLOPS_*`)
- If you don‘t know the runtime contribution of code regions/functions
  -> Use `gprof` or `perf` to create runtime profile
- Select Top3 functions for instrumentation
  - Short region runtime → instrumentation at coarser level
  - Long region runtime (>30s) → instrumentation at finer level
- Run application with instrumentation (`L2,L3,MEM,DATA,FLOPS_*`)

- Compare not time-based metrics:
  - How much of my overall L2 traffic is caused by this code region
  - How many FP instructions are executed in the region compared to the base run

NHR FAU

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Performance analysis of Jacobi 2D 5pt stencil (double-precision)

Jacobi 2D 5pt stencil (DP)

Appropriate performance metric:

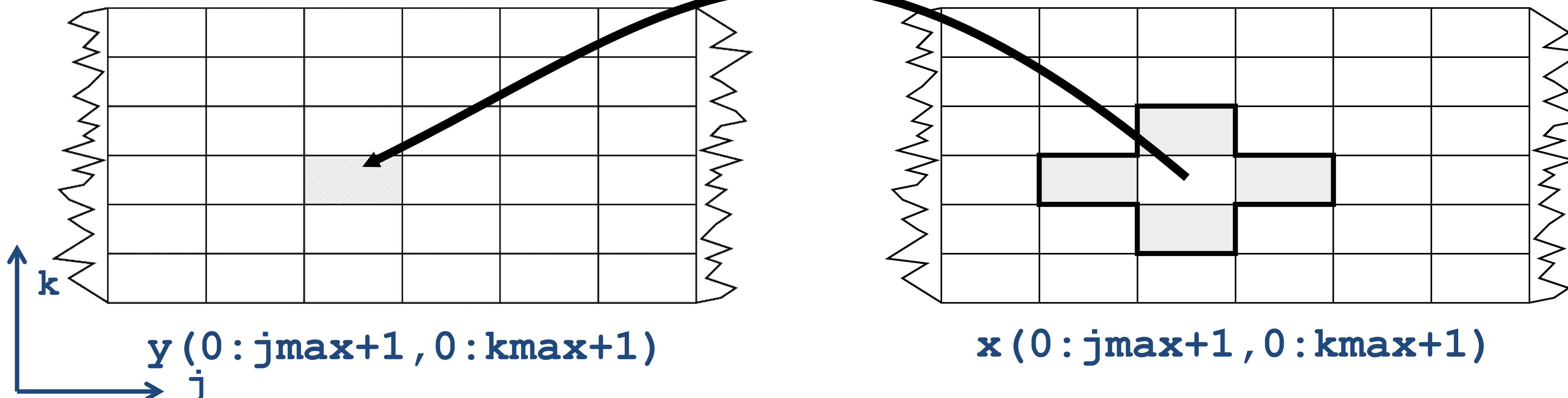
“Lattice site updates per second” [LUP/s]

(here: Multiply by 4 FLOP/LUP to get FLOP/s rate)

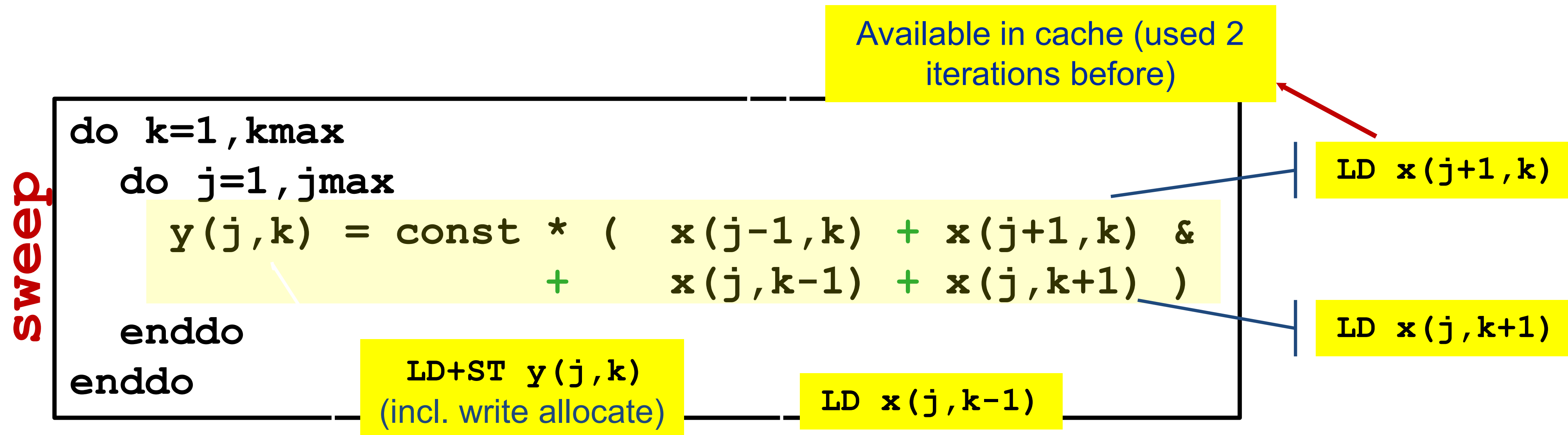
```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (  x(j-1,k) + x(j+1,k) &
                     +   x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

sweep

2 arrays



Jacobi 2D 5pt stencil (DP)



Naive balance (incl. write allocate):

`x( :, :)` : **3 LD +**

`y( :, :)` : **1 ST+ 1LD**

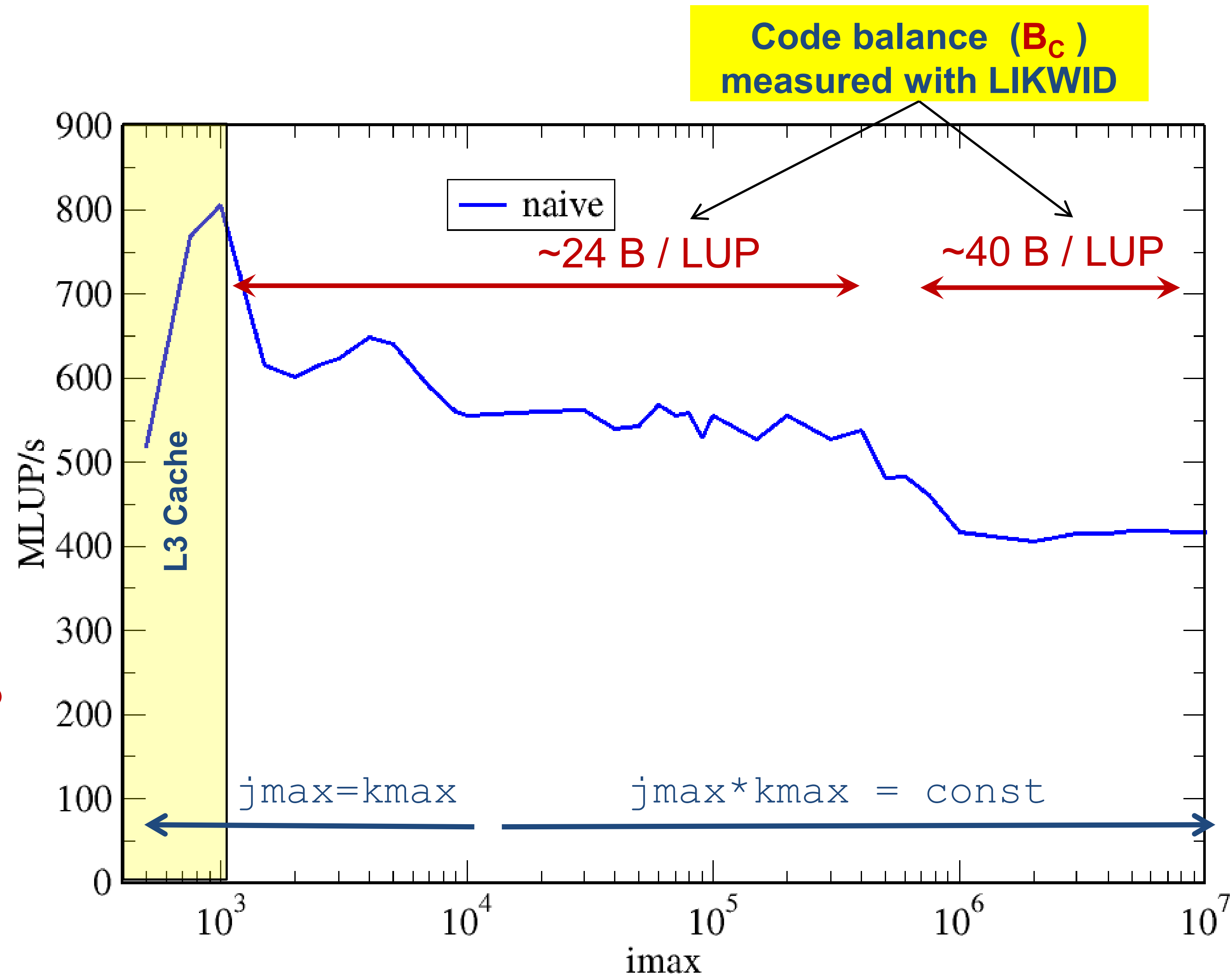
Reminder "write allocate":
If a cache line is not present in L1 cache at a store, the CL needs to be loaded into L1 first.

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

Jacobi 2D 5pt stencil (DP)

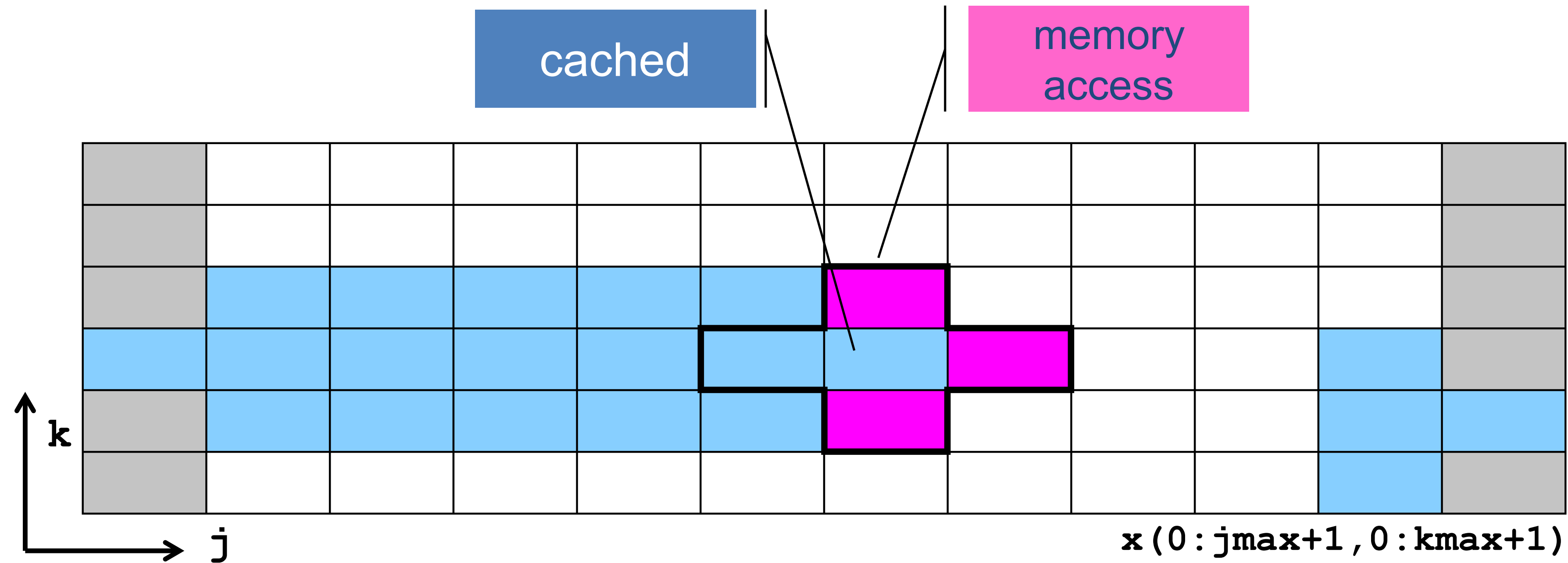
Questions:

1. **How to achieve 24 B/LUP also for large `jmax`?**
2. **How to sustain > 600 MLUP/s for `jmax>10⁴` ?**
3. **Why 24 B/LUP anyway??** **Naïve balance was 40B/LUP!**



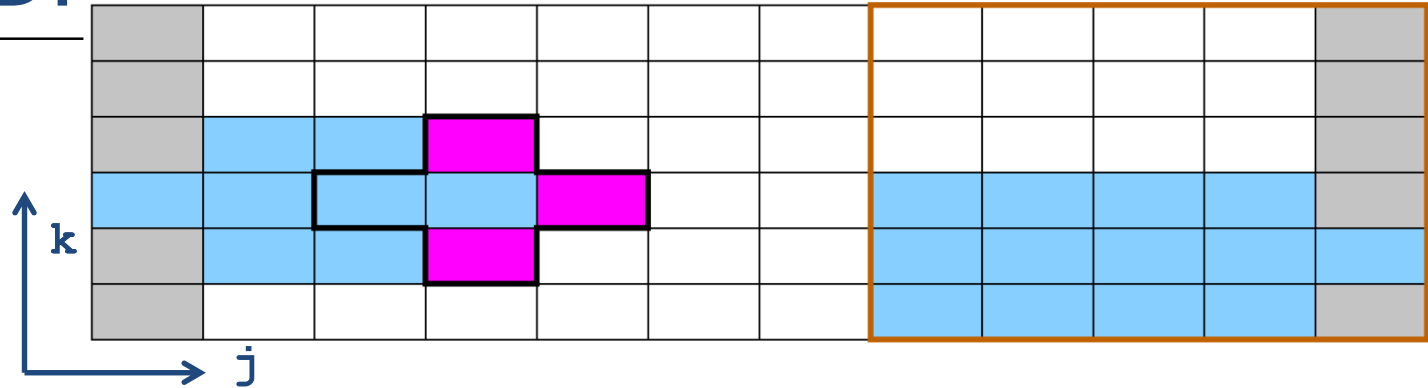
Intel Xeon E5-2690 v2 @ 3 GHz, ifort V13.1, L1: 32kB, L2: 256kB, L3: 25 MB

Jacobi 2D 5pt stencil (DP)



Worst case: Cache not large enough to hold 3 layers (rows) of grid (+assume „Least Recently Used" replacement strategy)

Jacobi 2D 5pt stencil (DP)

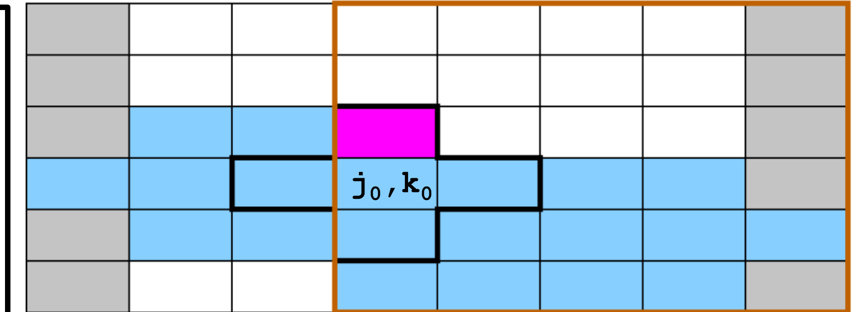


Reduce inner (j-) loop dimension successively

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

Jacobi 2D 5pt stencil (DP)

```
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
```



3 * jmax * 8B < CacheSize/2

"Layer condition"

3 rows of **jmax**

double precision

Safety margin (Rule of thumb)

Layer condition:

- No impact of outer loop length (kmax)
- No strict guideline (cache associativity – data traffic for **y()** not included)
- Needs to be adapted for other stencils, e.g., in 3D, long-range, multi-array,…

Jacobi 2D 5pt stencil (DP)

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

"Layer condition" fulfilled?

YES

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

NO

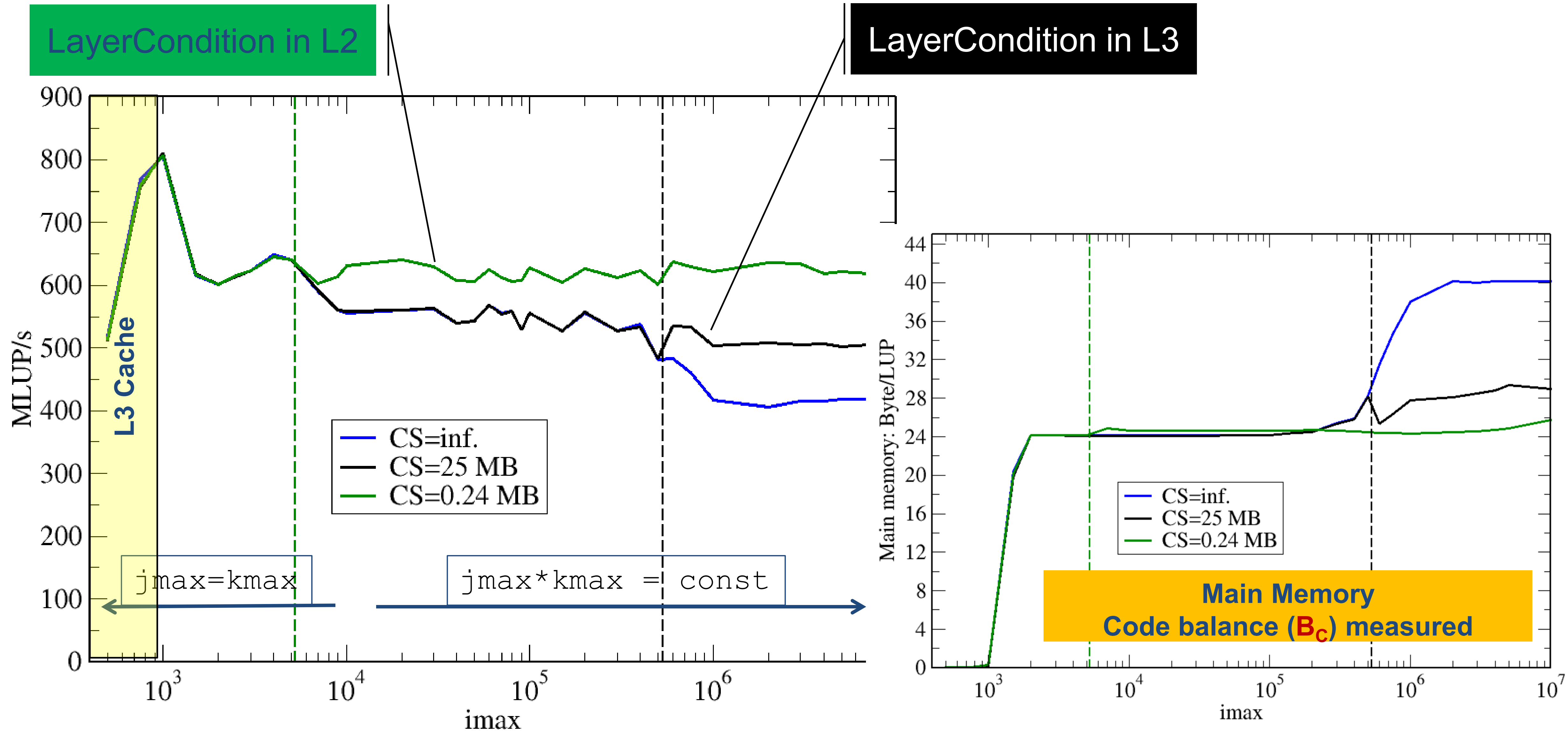
y: (1 LD + 1 ST) / LUP

```
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) with blocked j-loop



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

Summary

- LIKWID tool suite provides set of day-by-day tools for HPC users
- System information and pinning for efficient usage of compute resources
- Hardware performance event measurements for all HPC-relevant systems
- Powerful instrumentation to regions of interest
- Nvidia GPU profiling capabilities (other accelerators planned)

- Think about your code, not just profile it
  → if model does not fit, you learn something
  → if it fits, you gain insight where and what to optimize