LIKWID Performance Tools

Thomas Gruber <Thomas.Gruber@fau.de>
Regionales Rechenzentrum Erlangen
Friedrich-Alexander University Erlangen-Nuernberg

Who are we?

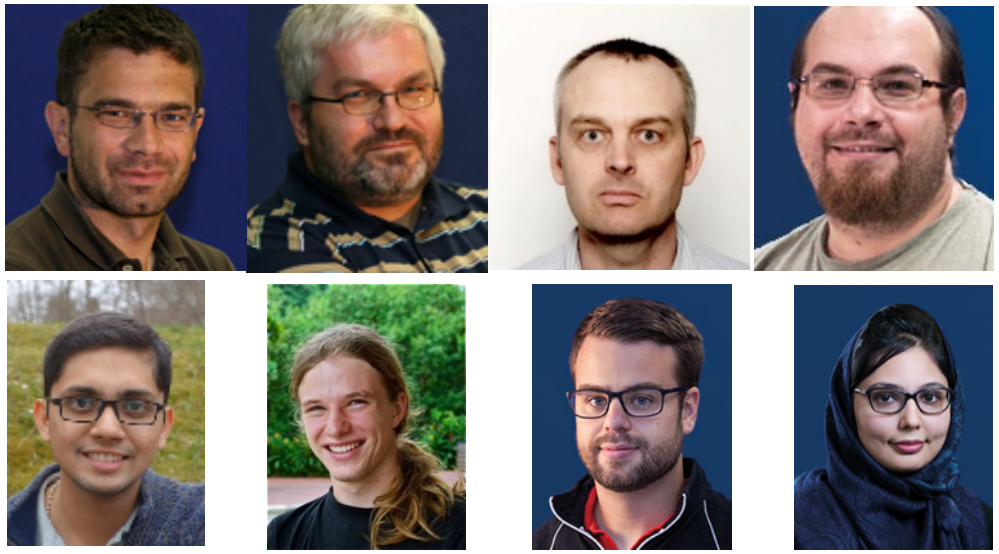
https://hpc.fau.de

- TIER 2 academic **HPC computing center** and **upcoming NHR center**


- 2 main and 4 special purpose systems
- In total 1656 nodes and 134 GPGPUs
- 4 people for software and support
- 2 system administrators

- Associated computer science **research group**
  - Performance Engineering
  - Performance Modeling
  - **Tool development**
  - Sparse and stencil solvers

LIKWID TOOLS
KERNCRAFT
OSACA



https://github.com/RRZE-HPC

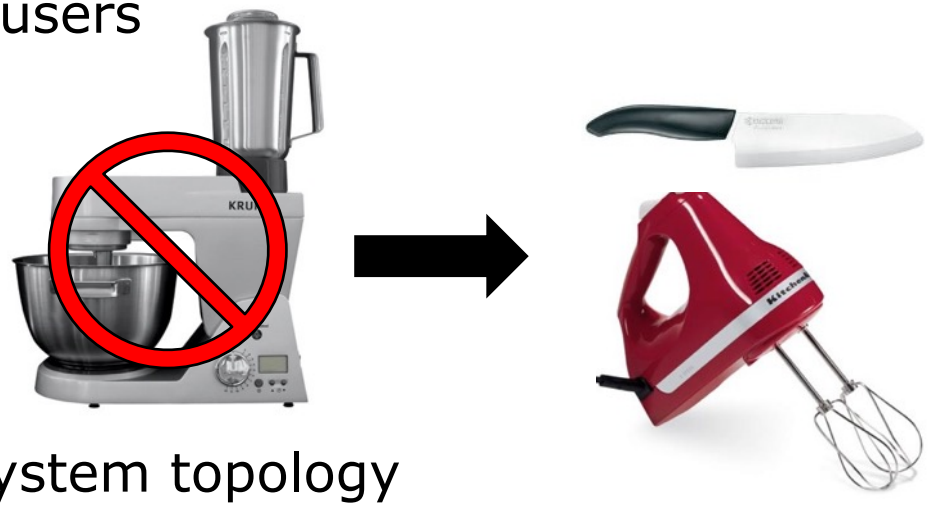
Tools for our and (maybe) your daily life

We perform a lot low-level analysis of hardware architectures and applications
What do we need?

- Read the hardware topology
- Flexible and comprehensive process/thread 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
- Control process and thread affinity according system topology
- Run micro-benchmarks to check system features
- Measure hardware events during application runs
- Determine energy consumption
- Manipulate CPU/Uncore frequencies
- Manipulate CPU features like prefetchers, …

LIKWID tools overview

- likwid-topology – Read topology of current system
- likwid-pin – Pin threads to CPU cores
- 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 tools on Goethe-HLR cluster

- VI-HPS workshop specific modules

```
module use /home/vihps/software/modulefiles
module load likwid
```

- Hands-On files

```
cp –r /home/vihps/public/likwid ~
```

- Job on Goethe-HLR

```
srun -N 1 -p general1 --reservation=VIHPS -t 2:00:00 --pty /bin/bash -l
```

Read system topology

```
$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
CPU type:       Intel Skylake SP processor
CPU stepping:   4
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                2
Cores per socket:       20
Threads per core:       2
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
[…]
```

Basic system layout

HW threads available in CPU set

Read system topology

```
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 40 ) ( 1 41 ) ( 2 42 ) ( 3 43 ) ( 4 44 ) ( 5 45 ) …
--------------------------------------------------------------------------------
Level:                  2
Size:                   1 MB
Cache groups:           ( 0 40 ) ( 1 41 ) ( 2 42 ) ( 3 43 ) ( 4 44 ) ( 5 45 ) …
--------------------------------------------------------------------------------
Level:                  3
Size:                   28 MB
Cache groups:           ( 0 40 1 41 2 42 3 43 4 44 5 …) ( 20 60 21 61 22 62 …)
```

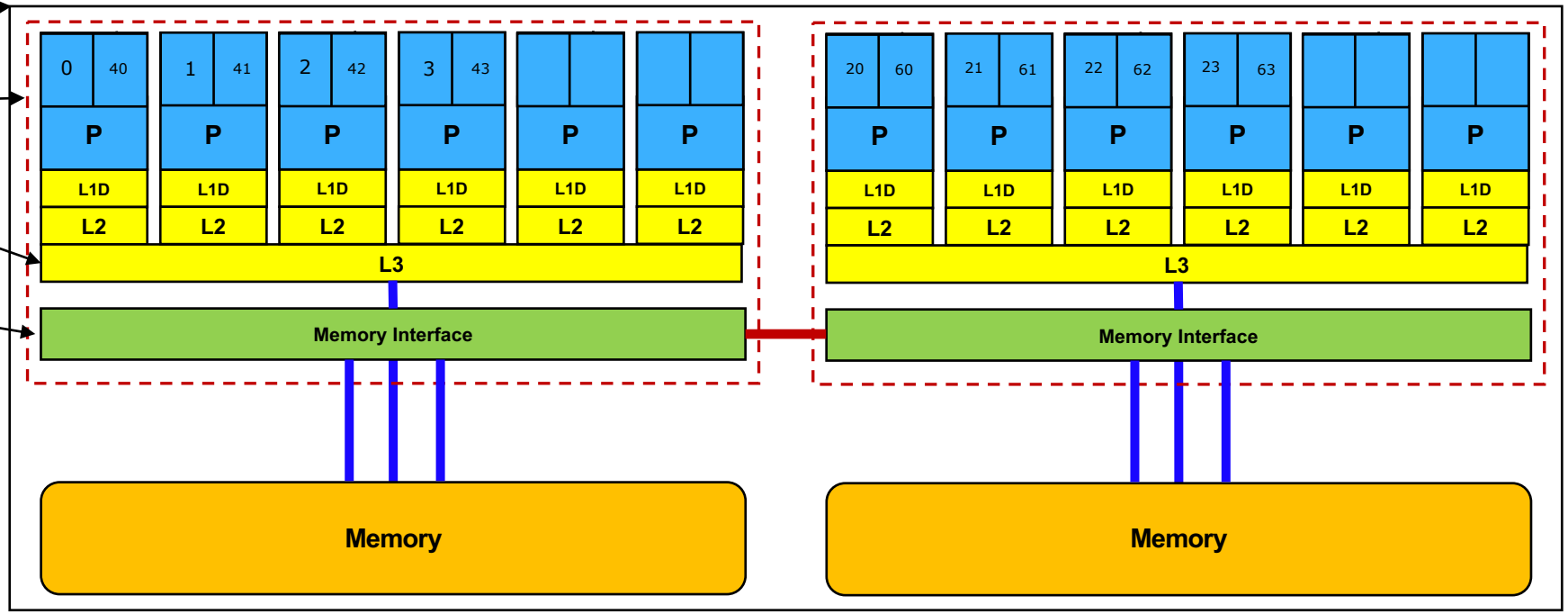
Read system topology

```
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:          2
--------------------------------------------------------------------------------
Domain:                0
Processors:            ( 0 40 1 41 2 42 3 43 4 44 5 45 6 46 7 47 8 48 9 49 10 50 …)
Distances:             10 21
Free memory:           90112.1 MB
Total memory:          95371 MB
--------------------------------------------------------------------------------
Domain:                1
Processors:            ( 20 60 21 61 22 62 23 63 24 64 25 65 26 66 27 67 28 68 …)
Distances:             21 10
Free memory:           94449.2 MB
Total memory:          96729 MB
--------------------------------------------------------------------------------
```

Control affinity of processes & threads

- Placement of SW threads crucial for performance
- `taskset -c 0,1 ./app`
- Task migrations should be avoided (cache trashing, reloads from memory, …)
- Complex systems group HW threads in affinity domains
  - Node
  - SocketX
  - Last-level CacheZ
  - MemoryY

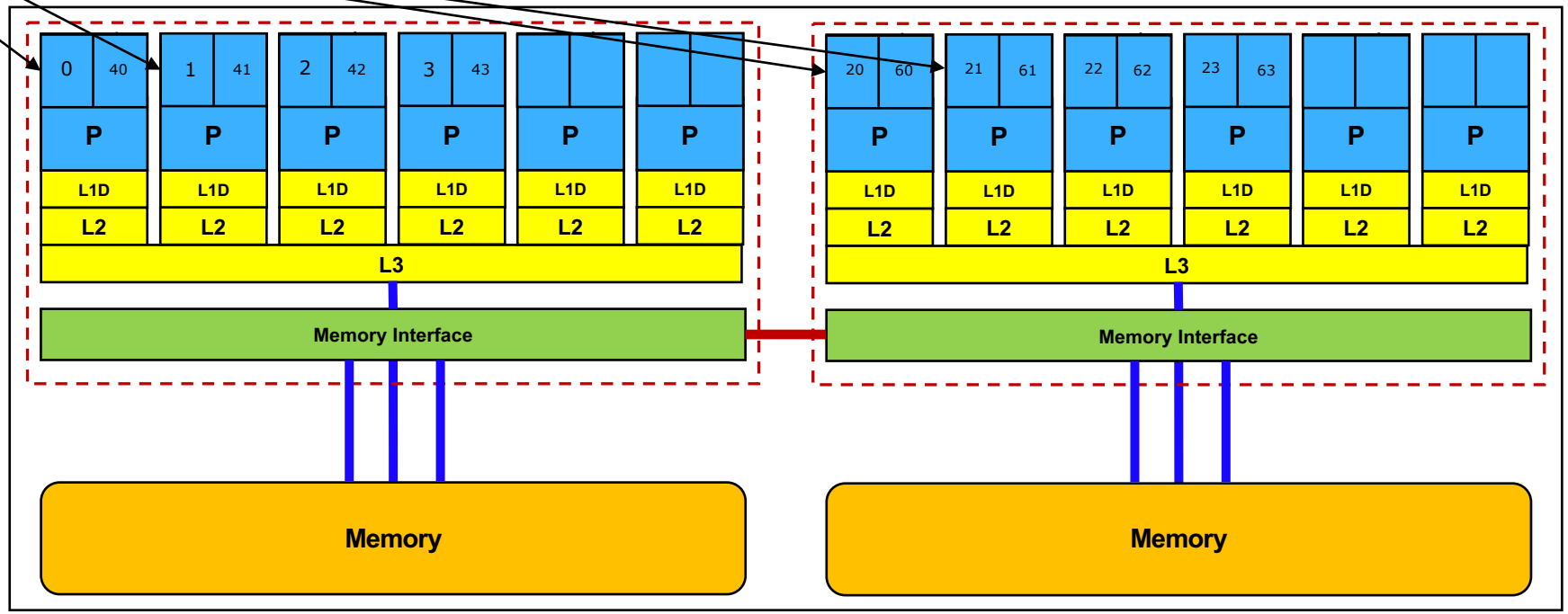
  (NUMA domain)



Control affinity of processes & threads

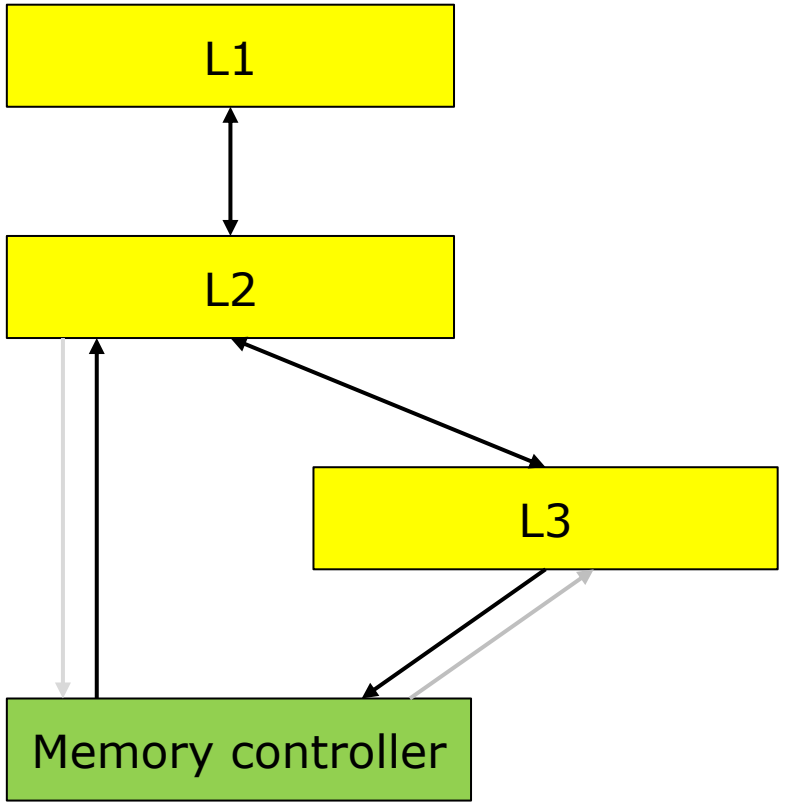
- `likwid-pin -c 0,1 ./app` == `taskset` with pinning
- `likwid-pin -c S0:0-1 ./app` `{0,1,2,…}`
  Domain S0, logical indicies 0,1 in „phys. HW threads first" sorted list
- `likwid-pin -c S0:0-1@S1:0-1 ./app`
  Combine selections

- Expression syntax:
  `E:M0:20:1:2`



RECAP: Cache hierarchy in Intel Skylake SP systems

- Intel Skylake SP system use an L3 victim cache with „clever" heuristics
- L2 loads data from memory directly
- L3 prefetcher may load data from memory (default off)
- Read-only data evicted from L2:
  - If it fits in L3 cache → evict to L3 cache
  - If it is larger than L3 cache → drop cache lines
- Modified data always evicted from L2 to L3
- No (?) write-backs to memory directly



Hardware performance monitoring

- Control performance measurements of x86, x86_64, ARM, POWER and Nvidia GPUs
- Measurement modes:
  - Start-to-End (whole application run)
  - Timeline mode (time-based sampling)
  - Stehtoscope mode (monitoring from the outside)
  - MarkerAPI (code instrumentation – CPU & GPU)
- `likwid-perfctr -c/-C <cpus> -g <cpuevents> (-m) (-t <time>) ./app`

With pinning

Activate MarkerAPIs

Time-based sampling

Hardware performance monitoring

```
$ likwid-perfctr -C 0 -g L3 hostname
--------------------------------------------------------------------------------
CPU name:        Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
CPU type:        Intel Skylake SP processor
CPU clock:       2.39 GHz
--------------------------------------------------------------------------------
node46-021.cm.cluster
--------------------------------------------------------------------------------
Group 1: L3
+----------------------+---------+------------+
|         Event        | Counter | HWThread 0 |
+----------------------+---------+------------+
|   INSTR_RETIRED_ANY  |  FIXC0  |       1207 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  |     489853 |
|  CPU_CLK_UNHALTED_REF |  FIXC2  |     344352 |
|    L2_LINES_IN_ALL   |   PMC0  |       6141 |
|     L2_TRANS_L2_WB   |   PMC1  |       3292 |
| IDI_MISC_WB_DOWNGRADE |   PMC2  |       1222 |
|  IDI_MISC_WB_UPGRADE |   PMC3  |          0 |
+----------------------+---------+------------+
```

Basic system info

Application output

Raw event table (one column per HW thread)

Events directly usable but not intuitive

Hardware performance monitoring

Metric	HWThread 0
Runtime (RDTSC) [s]	0.0020
Runtime unhalted [s]	0.0002
Clock [MHz]	3400.1931
CPI	405.8434
L3 load bandwidth [MBytes/s]	196.5215
L3 load data volume [GBytes]	0.0004
L3 evict bandwidth [MBytes/s]	0
L3 evict data volume [GBytes]	0
L3|MEM evict bandwidth [MBytes/s]	105.3491
L3|MEM evict data volume [GBytes]	0.0002
Dropped CLs bandwidth [MBytes/s]	39.1059
Dropped CLs data volume [GBytes]	0.0001
L3 bandwidth [MBytes/s]	301.8706
L3 data volume [GBytes]	0.0006

If used with performance group:

Table with derived metrics (one column per HW thread)

If multiple threads, also statistics tables are printed

Enable CSV output with `-O`

Searching for bottlenecks?

- Use a function profiler like gprof, xray, perf, …
- For GCC use **`-pg`** and **`gprof <exec> gmon.out`**

Time(%)	Self(%)	Call count	Function	File:line
74.21	74.21	100	runLoop	matrix.c:50
22.36	22.36	1	time_init	timer.c:97
3.40	3.40	3	fillMatrix	matrix.c:26
100.00	0.03	1	main	matrix.c:71

So, how to restrict measurements to the runLoop function?

Hardware performance monitoring (Code instrumentation)

```
#include <likwid-marker.h>
int main(int argc, char* argv[]) {
LIKWID_MARKER_INIT;
#pragma omp parallel {LIKWID_MARKER_REGISTER("omp
#pragma omp parallel {LIKWID_MARKER_START("ompca
for (int i = 0; i < iters; i++) {
    external_omp_call(size, A, B, C, D);
}
#pragma omp parallel {LIKWID_MARKER_STOP("ompcal
LIKWID_MARKER_CLOSE;
}
```

- $CC -I$LIKWID_INCDIR
  -L$LIKWID_LIBDIR
  -DLIKWID_PERFMON
  … -llikwid

```
$ likwid-perfctr -C S0:0-1 -g FLOPS_DP -m ./app
Region ompcall, Group 1: FLOPS_DP
+-------------------+------------+------------+
|    Region Info    | HWThread 0 | HWThread 1 |
+-------------------+------------+------------+
| RDTSC Runtime [s] |   0.575534 |   0.575600 |
|     call count    |         10 |         10 |
+-------------------+------------+------------+
+----------------------+------------+------------+
|        Metric        | HWThread 0 | HWThread 1 |
+----------------------+------------+------------+
|  Runtime (RDTSC) [s] |     0.5755 |     0.5756 |
| Runtime unhalted [s] |     0.5972 |     0.5974 |
|      Clock [MHz]     |  2959.7654 |  2959.7348 |
|          CPI         |     1.1910 |     1.1907 |
|     DP [MFLOP/s]     |   596.5403 |   595.8154 |
|   AVX DP [MFLOP/s]   |          0 |          0 |
|  AVX512 DP [MFLOP/s] |          0 |          0 |
|   Packed [MUOPS/s]   |   298.2701 |   297.9076 |
|   Scalar [MUOPS/s]   |     0.0002 |     0.0002 |
|  Vectorization ratio |    99.9999 |    99.9999 |
+----------------------+------------+------------+
```

Hardware performance monitoring (MPI+X applications)

- `$ likwid-mpirun -np 2 -t 10 ./app`
  Two MPI processes with 10 OMP threads each
- `$ likwid-mpirun -nperdomain M:1 -t 10 ./app`
  One MPI process per NUMA domain with 10 OMP threads
- `$ likwid-mpirun -np 2 -g FLOPS_DP (-m) ./app`
  Measure FLOPS_DP on two MPI processes (with MarkerAPI)

- Tool for interactive measurement of applications
  Use instead of srun, mpirun or mpiexec
- MarkerAPI for GPUs supported

LIKWID Performance Tools

- Dense DP matrix-vector-multiplication
  - Derive Roofline Model with LIKWID

Dense DP matrix-vector-multiplication

- Common operation in HPC codes
- Simple code but how does it map to hardware?

```
do c = 1 , C
  tmp=x(c)
  do r = 1 , R
    y(r) = y(r) + A(r,c) * tmp
  enddo
enddo
```

Right hand side (RHS) vector

Left hand side (LHS) vector

- Code balance (inner loop):
  - 2 LD, 1 ST
  - 1 ADD, 1 MULT

$$B_c = \frac{Data\ Transfers\ (Byte)}{Work\ Metric\ (flops, CLupdates, \dots)} = \frac{3\ words}{2\ flops} = \frac{24\text{B}}{2\text{F}}$$

B_c can be calculated for each memory hierarchy level

- RHS only loaded once per outer loop
- What can we do to optimize the code?

Dense DP matrix-vector-multiplication

```
#pragma omp parallel private(k)
{

  for (k = 0; k < ROUNDS; k++) {
    #pragma omp for private(j)
    for (i = 0; i < R; i++)
    {
      for (j = 0; j < C; ++j) {
        cvec[j] += mat[offset+j] * bvec[i];
      }
    }
  }
}
```

- Compile with `-DLIKWID_PERFMON`
- Link with LIKWID library (`-llikwid`)
- `LIKWID_MARKER_REGISTER()` recommended

```
#include <likwid-marker.h>
LIKWID_MARKER_INIT;
#pragma omp parallel private(k)
{
  LIKWID_MARKER_START("dMVM")
  for (k = 0; k < ROUNDS; k++) {
    #pragma omp for private(j)
    for (i = 0; i < R; i++)
    {
      for (j = 0; j < C; ++j) {
        cvec[j] += mat[offset+j] * bvec[i];
      }
    }
  }
  LIKWID_MARKER_STOP("dMVM")
}
LIKWID_MARKER_CLOSE;
```

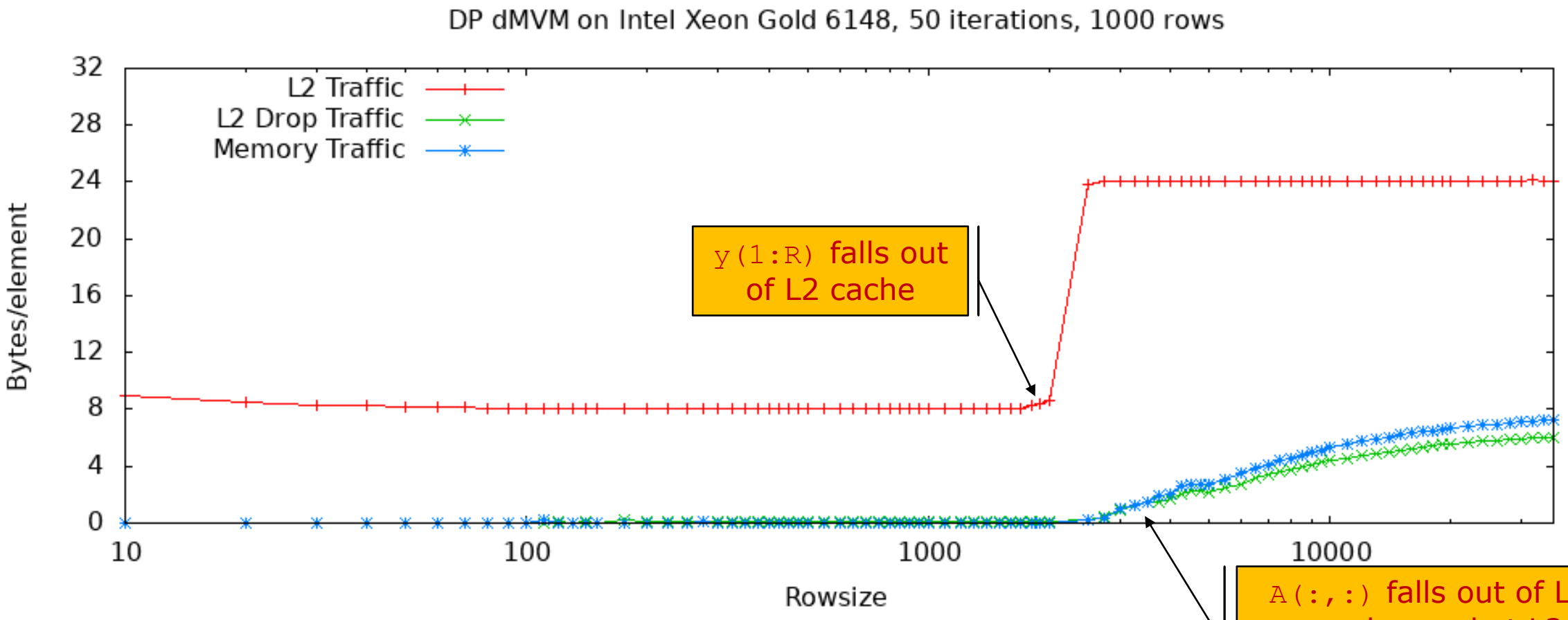
Dense DP matrix-vector-multiplication

- x(c) in register during inner loop
- A(:,:) no reuse
  → loaded from memory
- y(:) frequently updated
  → likely to stay in cache

```
do c = 1 , C
  tmp=x(c)
  do r = 1 , R
    y(r) = y(r) + A(r,c) * tmp
  enddo
enddo
```

- With increasing R, y(:) might be too large for innermost cache
  → increased data traffic from lower cache levels

Dense DP matrix-vector-multiplication



Dense DP matrix-vector-multiplication

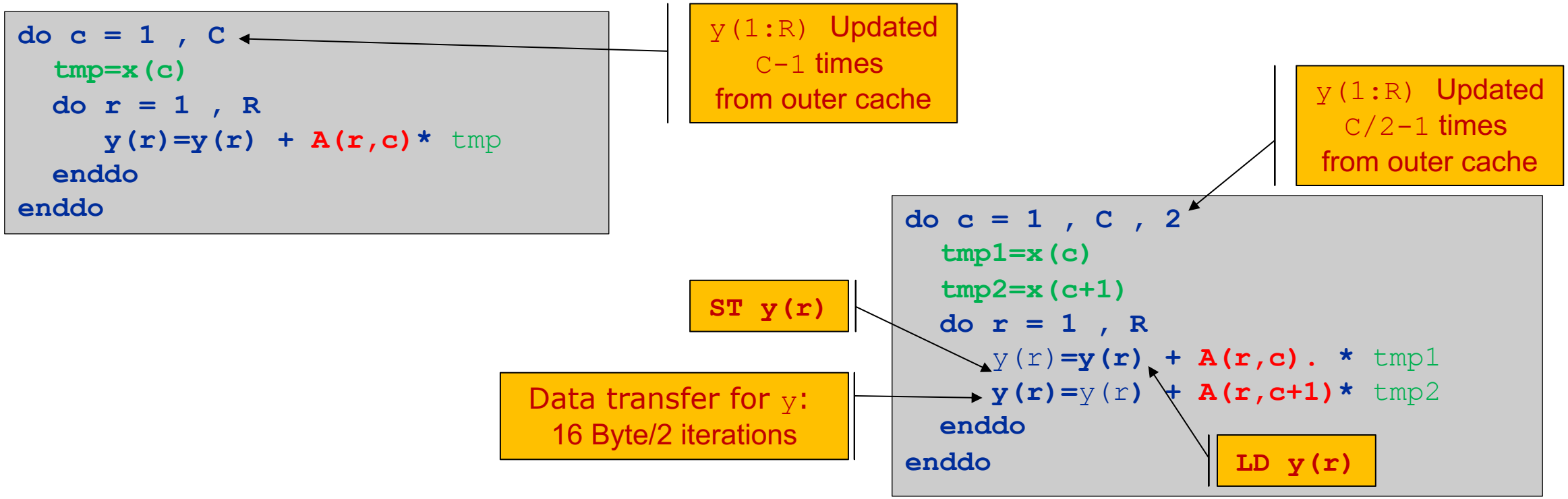
Attempt 1: Support vectorization with 8-*way inner loop unrolling*

- Unroll loop by 8 to help compiler vectorize with AVX512
- R multiple of 8 or reminder loop required
- Register of `tmp` holds 8 values (AVX512, DP)

```
do c = 1,C
  tmp=x(c)
  do r = 1,R,4   ! R is multiple of 4
     y(r)   = y(r)   + A(r,c)   * tmp
     y(r+1) = y(r+1) + A(r+1,c)* tmp
     y(r+2) = y(r+2) + A(r+2,c)* tmp
     y(r+3) = y(r+3) + A(r+3,c)* tmp
  enddo
  do q = r,R
     y(r)   = y(r)   + A(r,c)   * tmp
  enddo
enddo
```

Dense DP matrix-vector-multiplication

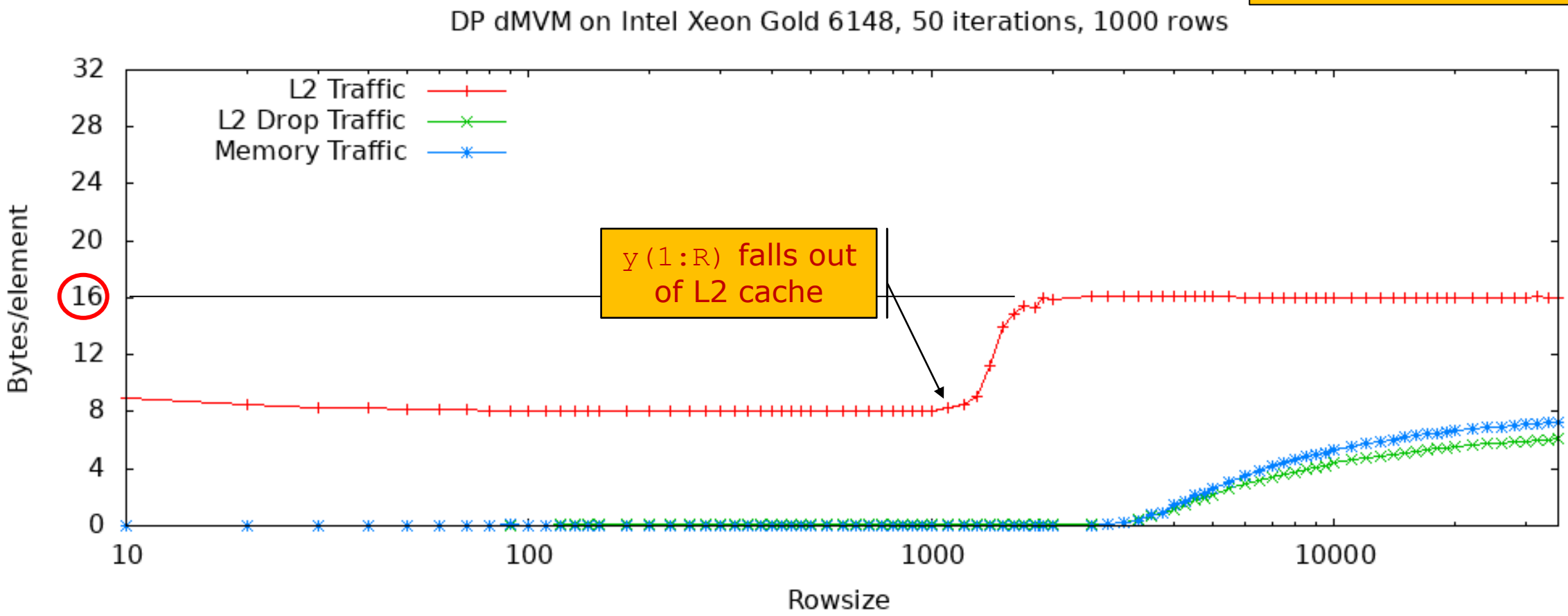
Attempt 2: Reducion of in-cache traffic *"2–way unroll & jam"*



Reduces worst case code balance to $B_C^i = (8+8)\ B/iteration$

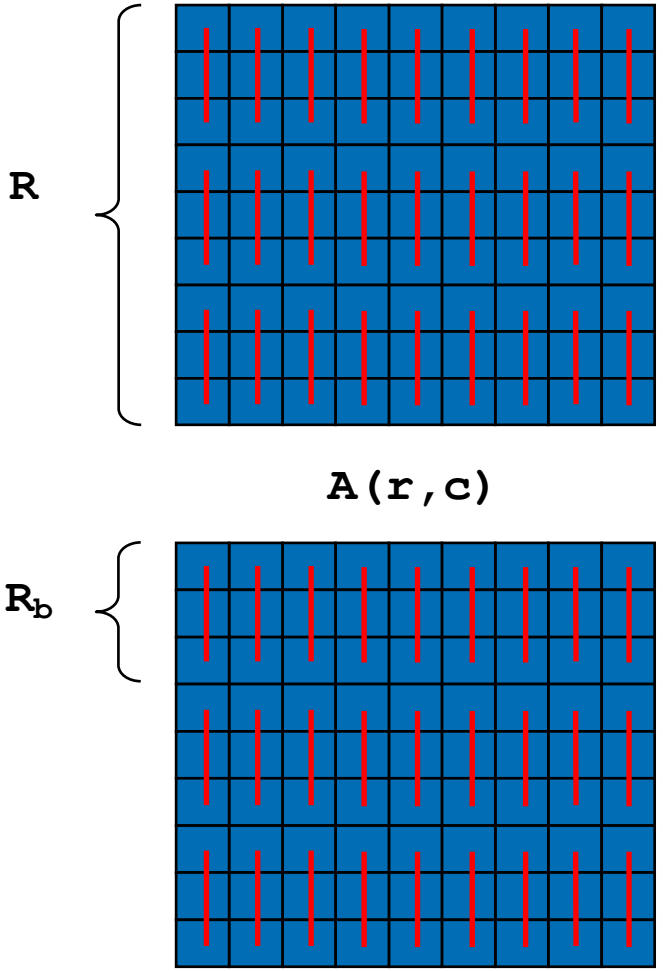
General for m-way unroll & jam: $B_C^i = (8+\frac{16}{m})\ B/iteration$

2–way unroll & jam



Dense DP matrix-vector-multiplication

Attempt 3: Reducion of cache traffic by *"blocking/tiling"*




```
do c = 1 , C
  tmp=x(c)
  do r = 1 , R
     y(r)=y(r) + A(r,c)* tmp
  enddo
enddo
```

```
do rb = 1 , R , Rb
 rbS = rb
 rbE = min((rb+Rb-1), R)
 do c = 1 , C
   do r = rbS , rbE
      y(r)=y(r) + A(r,c)*x(c)
   enddo
 enddo
enddo
```

R_b free parameter – adapted to caches size (commonly size/2)

`y(:)` may not fit into some cache → more traffic for lower level

`y(rbS:rbE)` may fit into some cache if R_b is small enough → traffic reduction

Dense DP matrix-vector-multiplication

Attempt 3: Reducion of cache traffic by *"blocking/tiling"*

- Blocking helps keeping LHS y(:) in cache
- Price: RHS x(:) loaded $^{R}/_{R_b}$ times

  → x(:) and y(:) contribute to worst case code balance

  - Original code:

$$\frac{16\mathrm{B}\cdot R\cdot C+8\mathrm{B}\cdot C}{R\cdot C\ iterations}=\left(16+\frac{8}{R}\right)\frac{\mathrm{B}}{it.}\cong 16\frac{\mathrm{B}}{it.}$$

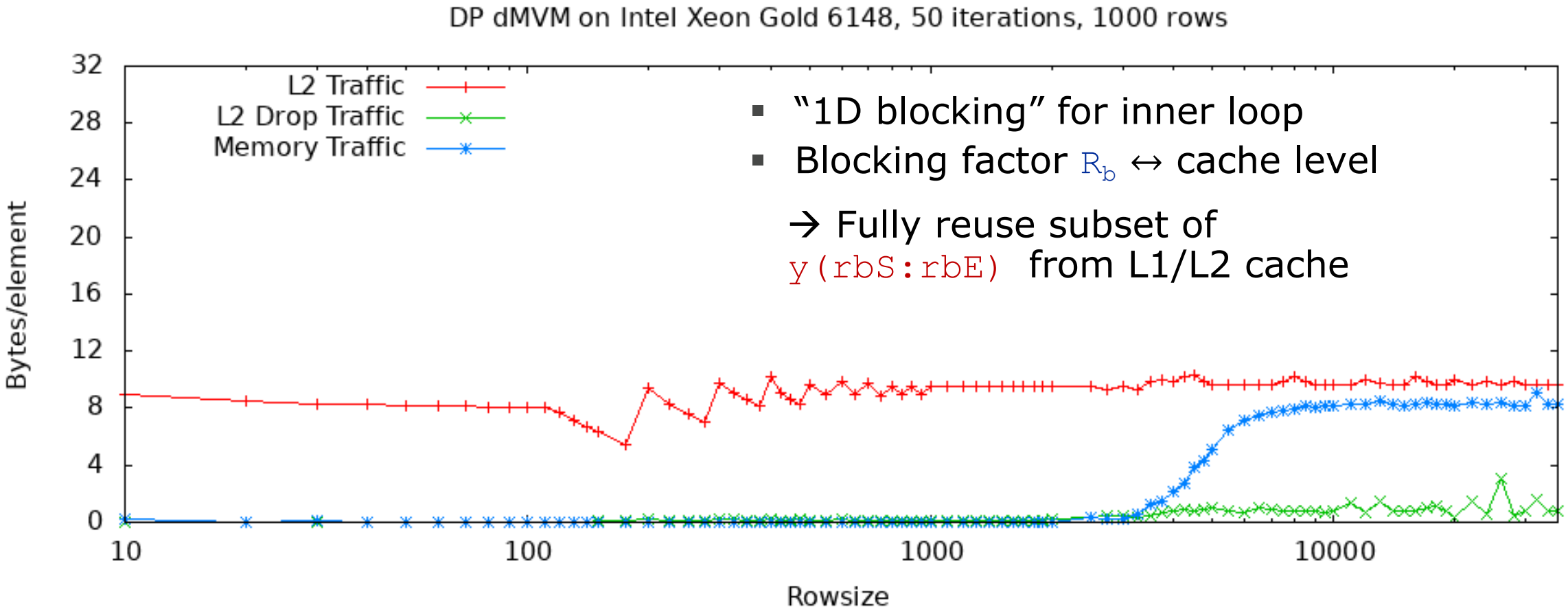
  - Blocking with factor R_b

$$\frac{16\mathrm{B}\cdot R+8\mathrm{B}\cdot C\cdot {}^{R}/_{R_b}}{R\cdot C\ iterations}=(\frac{16}{C}+\frac{8}{R_b})\frac{\mathrm{B}}{it.}\cong\frac{8}{R_b}\frac{\mathrm{B}}{it.}$$

→ Do not choose $\mathrm{R_b}$ too small

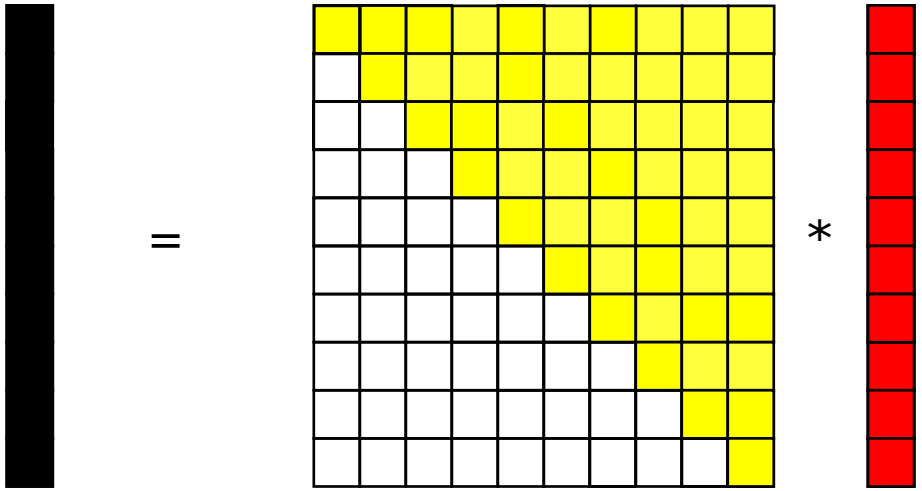
Dense DP matrix-vector-multiplication

Attempt 3: Reducion of cache traffic by *"blocking/tiling"*



Dense DP matrix-vector-multiplication

- But what if our input matrix is symmetric?
  - Let's use only the triangular matrix
    - Half the FP ops
    - Loading only half of the matrix




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

Dense DP matrix-vector-multiplication

Separate run per group

```
$ likwid-perfctr -C S0:0-2 -g <L2|L3|MEM> -m ./matrix_likwid
-------------------------------------------------------------
CPU name:     Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
CPU type:     Intel Skylake SP processor
CPU clock:    2.39 GHz
-------------------------------------------------------------
```

Compiled with
ROWSIZE=8000
COLSIZE=8000
ROUNDS=50

HW thread 0 uses most data

```
Region dMVM, Group 1: <L2|L3|MEM>
+-------------------------------+-----------+-------------+-------------+
|             Metric            | HWThread 0 | HWThread 1 | HWThread 2 |
+-------------------------------+-----------+-------------+-------------+
|     L2 data volume [GBytes]   |    14.3432 |     8.6370 |      1.9049 |
|     L3 data volume [GBytes]   |    11.6536 |     6.6859 |      2.6035 |
| Memory data volume [GBytes]   |    12.1111 |          0 |           0 |
+-------------------------------+-----------+-------------+-------------+
```

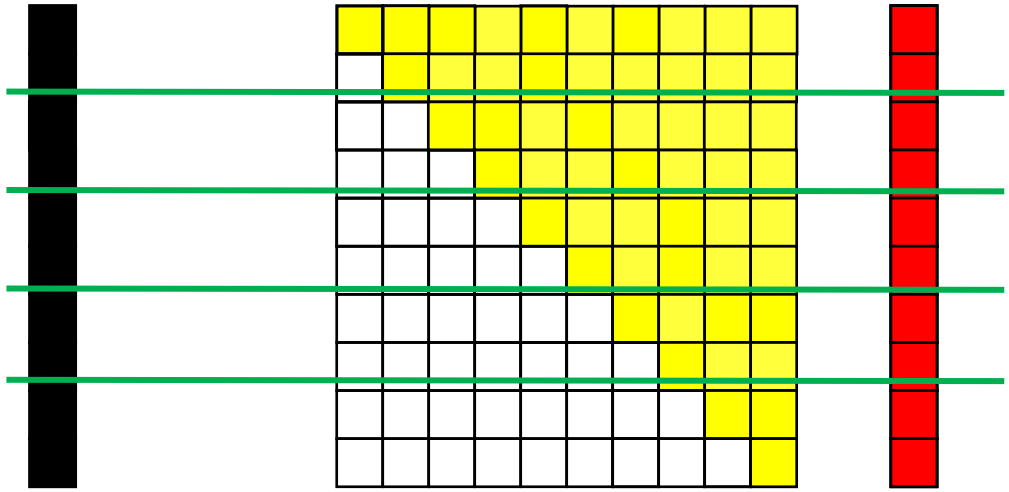
Memory measurements per CPU socket

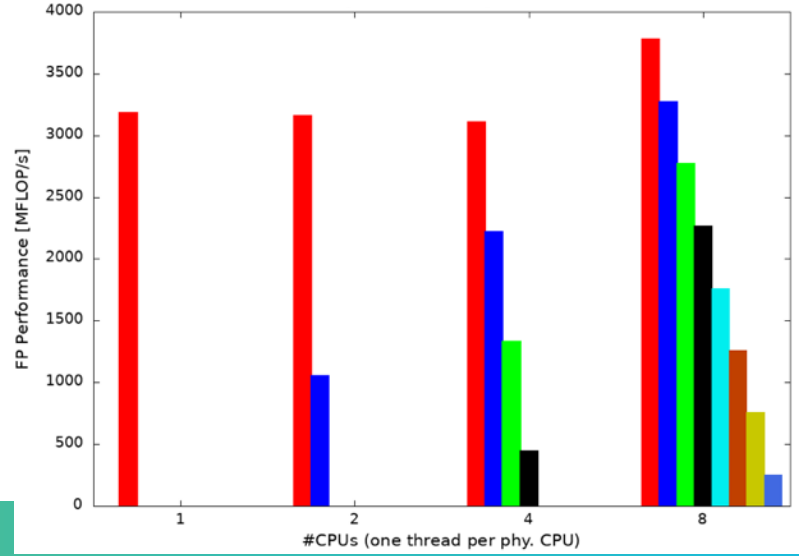
Dense DP matrix-vector-multiplication

- How to fix the load imbalance?

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

Implicit OpenMP barrier (busy waiting then sleep)



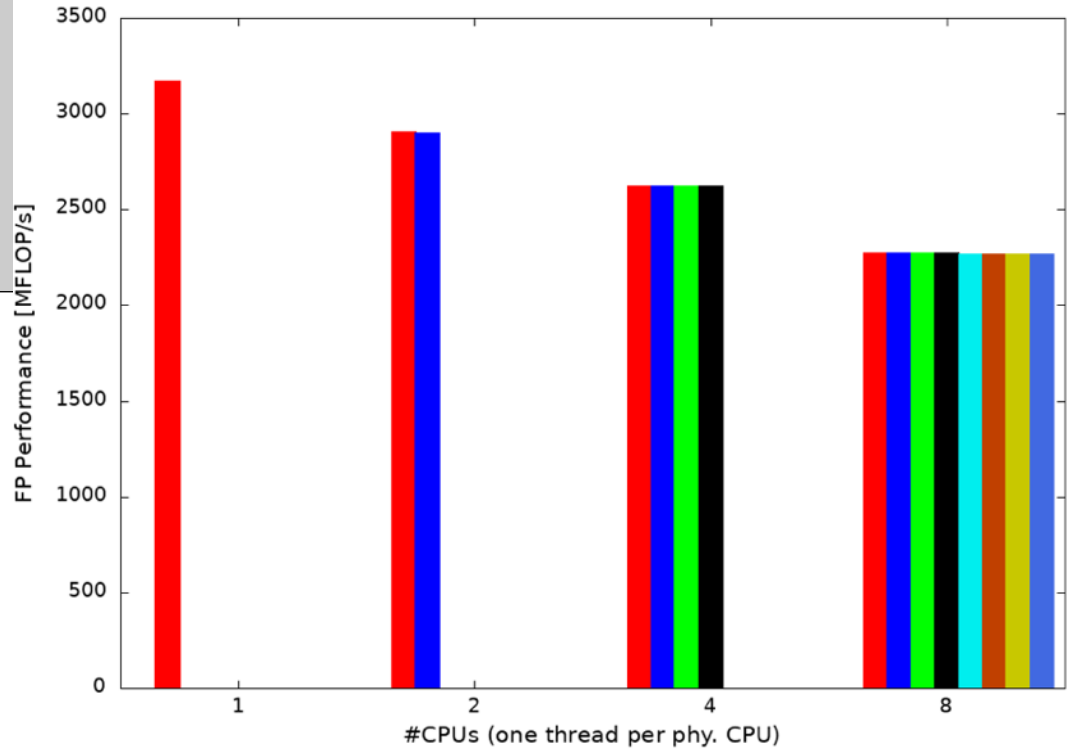


Dense DP matrix-vector-multiplication

- OpenMP schedule to the rescue!

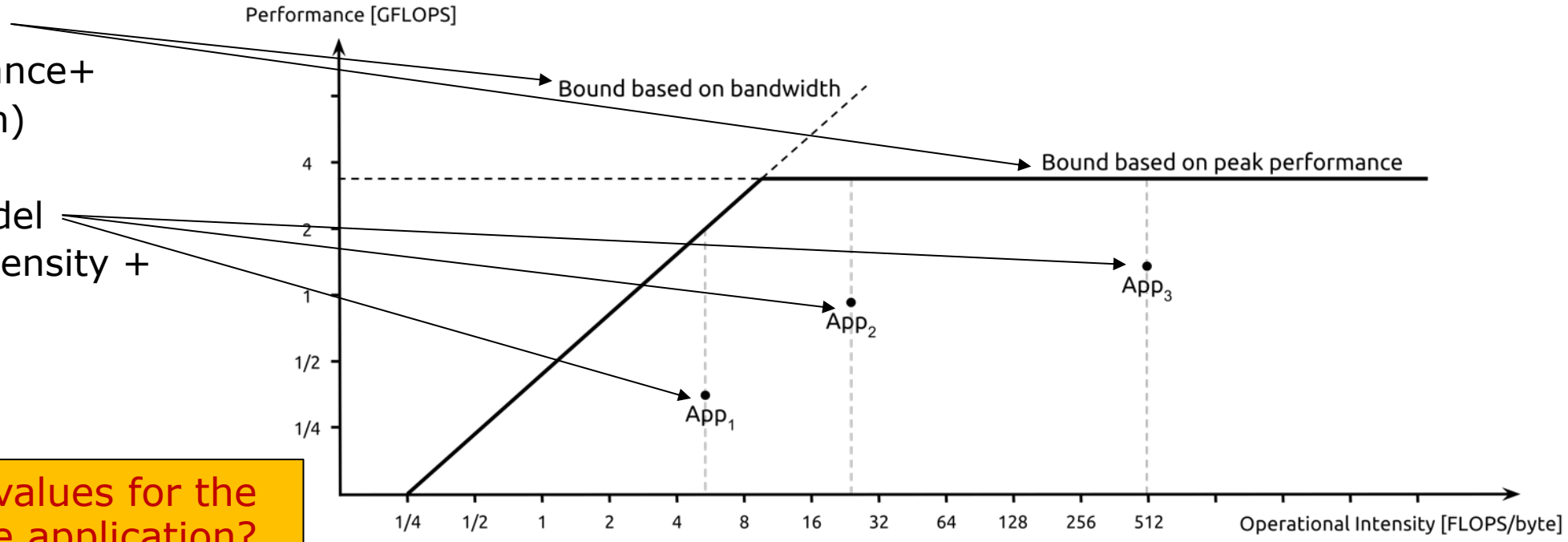
```
!$OMP PARALLEL DO PRIVATE(r) SCHEDULE(DYNAMIC)
do c = 1 , SIZE
  do r = 1 , c
     y(r) = y(r) + A(r,c)* x(c)
  enddo
enddo
!$OMP END PARALLEL DO
```

- Fill wait time with work
- Slower for fully regular problems (dyn. chunk calculation)



Empirical Roofline Model with LIKWID

- The Roofline Model is a well-known and easily descriptive performance model
- The model consists of two parts:
  - Machine model (peak performance+ peak bandwidth)
  - Application model (operational intensity + FLOP rate)



How to get the values for the machine and the application?

https://en.wikipedia.org/wiki/Roofline_model

Empirical Roofline Model with LIKWID

- Determine Machine Model
  - Calculate peak FLOP rate:
    <num_cores> × <op_width> × <num_ops_per_cycle> × <num_fma> × <cpu_freq>
  - Calculate peak memory bandwidth:
    <num_sockets> × <num_channels> × <mem_freq> × <num_controllers> × <data_transfer_size>

No guarantees for the formulas

- Problem: Theoretical values don't reflect reality
  - Lower cpu_freq with higher op_width
  - Protocol overhead not accounted properly
  - …

Empirical Roofline Model with LIKWID

- Determine Machine Model
  - Measure maximal FLOP rate:
    - `likwid-bench` provides `peakflops` benchmarks
    - Linpack/HPL
    - DGEMM

```
$ likwid-bench -t peakflops -W N:400kB:40:1:2
MFlops/s: 244182.88
$ likwid-bench -t peakflops_sse -W N:400kB:40:1:2
MFlops/s: 483947.04
$ likwid-bench -t peakflops_avx -W N:400kB:40:1:2
MFlops/s: 811089.34
$ likwid-bench -t peakflops_avx_fma -W N:400kB:40:1:2
MFlops/s: 1622287.83
$ likwid-bench -t peakflops_avx512_fma -W N:400kB:40:1:2
MFlops/s: 2705627.72
```

  - Measure maximal memory bandwidth:
    - STREAM benchmark
    - TheBandwidthBenchmark
    - likwid-bench provides kernels for common access patterns (copy, update, ddot, daxpy, stream triad, vector triad)

```
$ likwid-bench -t load_avx -W N:4GB:40:1:2
MByte/s: 204651.06
$ likwid-bench -t copy_avx512 -W N:4GB:40:1:2
MByte/s: 124753.85
```

Use a access pattern that reflects your application to get a meaningful model

Empirical Roofline Model with LIKWID

- Determine Application model
  - Limit measurements to region of interest (code instrumentation)
  - Use `likwid-perfctr -m` with groups `MEM_DP` or `MEM_SP` (`STAT` table)

```
$ likwid-perfctr -C S0:0-19 -g MEM_DP -m ./app
+--------------------------------------+------------+-----------+--------------+--------------+
|                Metric                |     Sum    |    Min    |      Max     |      Avg     |
+--------------------------------------+------------+-----------+--------------+--------------+
|           Clock [MHz] STAT           | 44074.0599 | 2159.4363 |    2593.7167 |    2203.7030 |
|           DP [MFLOP/s] STAT          |  3953.8828 |  167.5192 |     201.1234 |     197.6941 |
|   Memory bandwidth [MBytes/s] STAT   | 94095.2691 |         0 |   94095.2691 |    4704.7635 |
|      Operational intensity STAT      |     0.0414 |    0.0018 |       0.0021 |       0.0021 |
+--------------------------------------+------------+-----------+--------------+--------------+
```

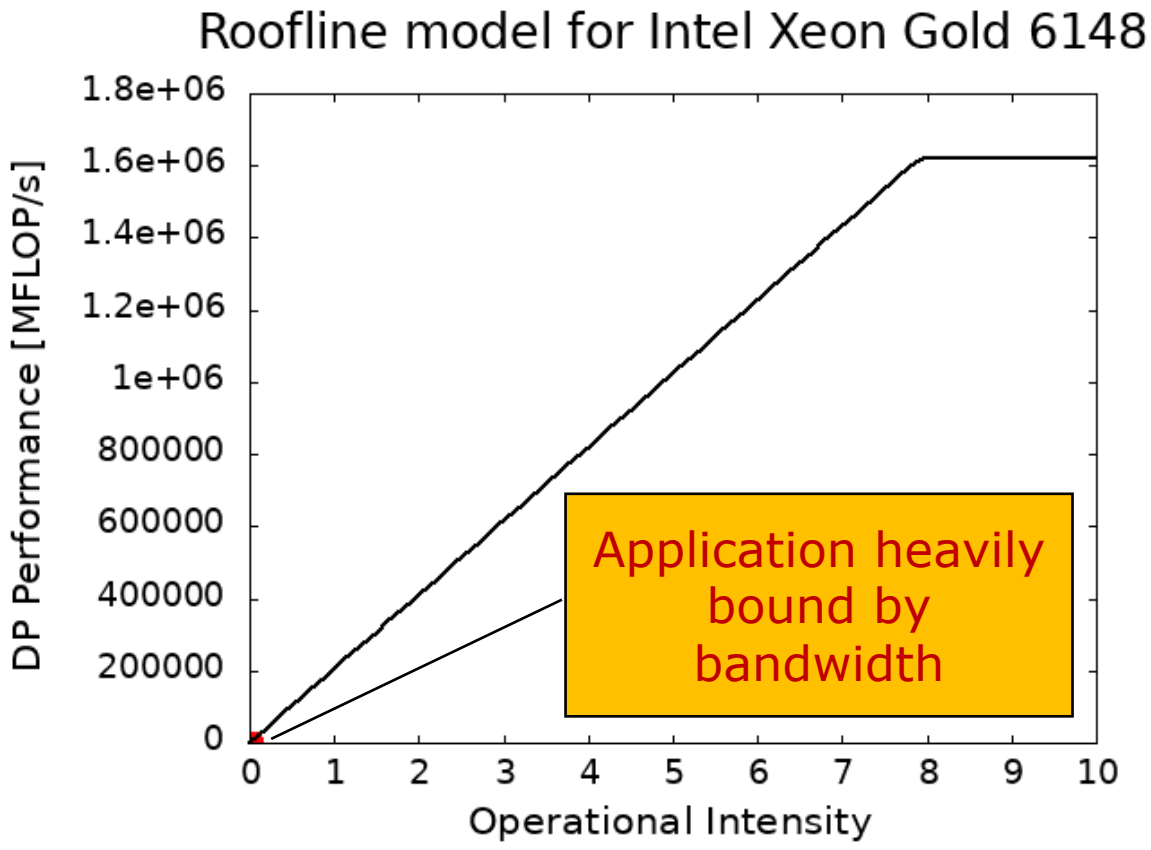
Empirical Roofline Model with LIKWID

- Draw Roofline Model (gnuplot)

```
maxperf = 1622287.83 # MFlops/s
maxband = 204651.06 # MByte/s
op_ins = 0.0414
app_perf = 3953.8828


set object circle at op_ins,app_perf \
  radius char 0.5 fc rgb 'red' fs solid


roof(x) = maxperf > (x * maxband) ? \
  (x * maxband) : maxperf


plot roof(x) notitle
```

Application model as red dot

Roof line function



Summary

- LIKWID is a whole tool suite for running and profiling apps
- Provide easy-to-use but sophisticated tools for the daily work
- Supports for all relevant CPU architectures (x86, x86_64, ARM, POWER)
- Supports Nvidia GPUs
- Main target: node-level performance engineering

Webpage: https://hpc.fau.de/research/tools/likwid/

Releases: https://ftp.fau.de/pub/likwid/

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