Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# LIKWID

## VI-HPS Tuning Workshop 2020, CINECA

Thomas Gruber
Regional Computing Center Erlangen (RRZE)
Friedrich-Alexander Universität Erlangen-Nürnberg



Sorry: next time VI-HPS style

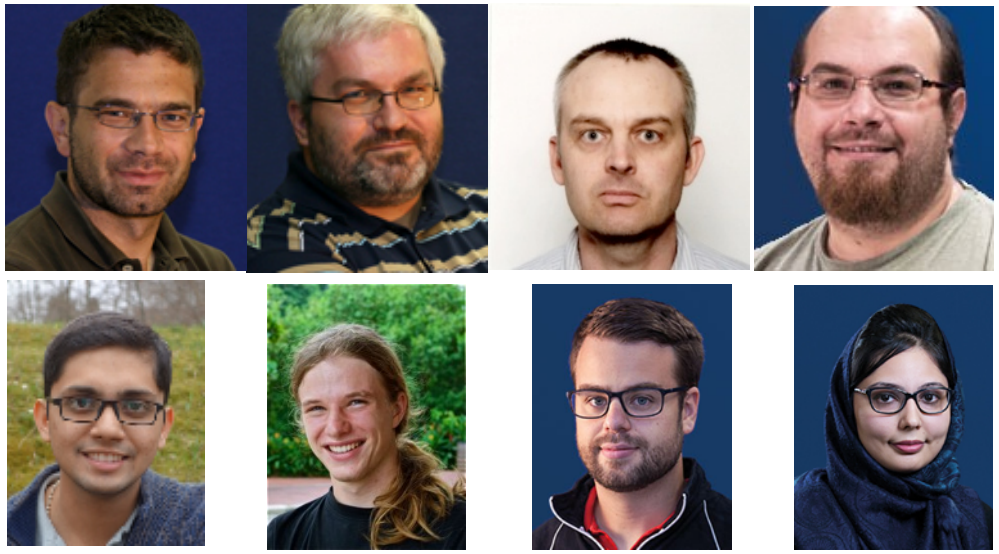
# Who are we? https://hpc.fau.de

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

- TIER 2 academic **HPC computing 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

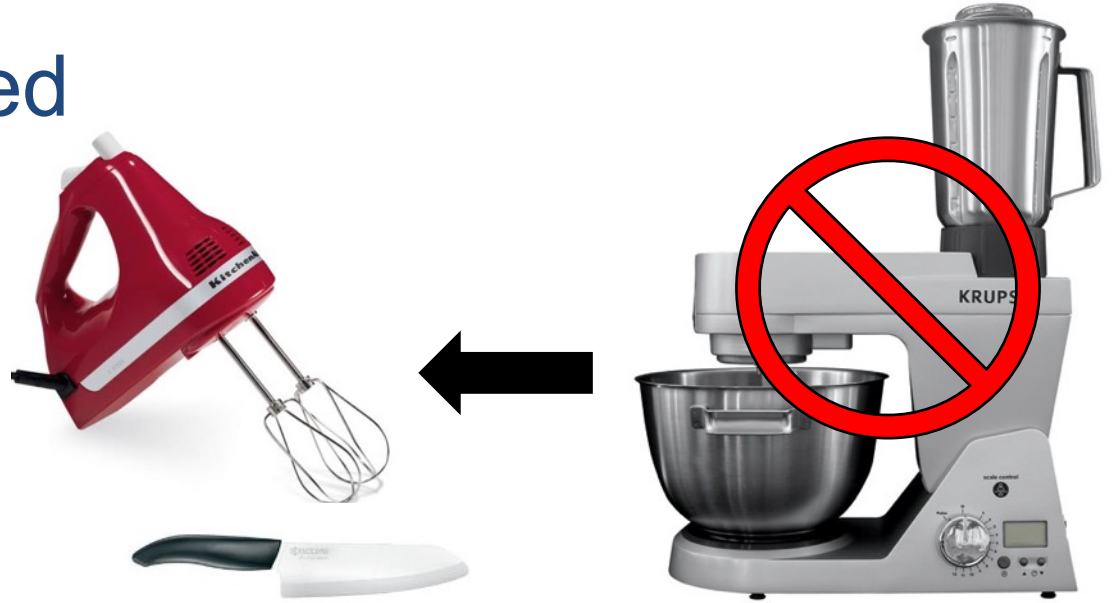
# Tools for the daily life

- 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 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-topology

```
$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       POWER9  altivec supported
CPU type:       POWER9 architecture
CPU stepping:   0
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                2
Cores per socket:       16
Threads per core:       4
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               1               0               0               *
2               2               0               0               *
3               3               0               0               *
4               0               1               0               *
5               1               1               0               *
6               2               1               0               *
7               3               1               0               *
```

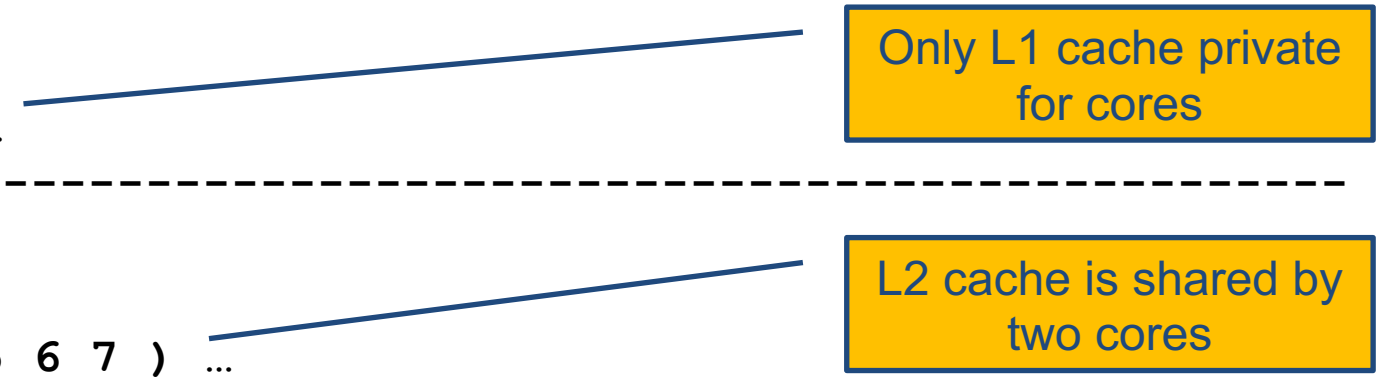
Basic system layout

HW threads 0 and 1 not distinct core

HW threads available in CPU set

# likwid-topology

```
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 1 2 3 ) …
--------------------------------------------------------------------------------
Level:                  2
Size:                   512 kB
Cache groups:           ( 0 1 2 3 4 5 6 7 ) …
--------------------------------------------------------------------------------
Level:                  3
Size:                   10 MB
Cache groups:           ( 0 1 2 3 4 5 6 7 ) …
```



# likwid-topology

```
*************************************************************
NUMA Topology
*************************************************************
NUMA domains:         6
-------------------------------------------------------------
Domain:               0
Processors:           ( 0 1 2 3 4 5 …)
Distances:            10 40 80 80 80 80
Free memory:          95479.6 MB
Total memory:         126804 MB
-------------------------------------------------------------
Domain:               252
Processors:           ( )
Distances:            80 80 10 80 80 80
Free memory:          16128 MB
Total memory:         16128 MB
```

NUMA node is equal to CPU socket

Empty NUMA domains for attached memory like HBM or GPU memory

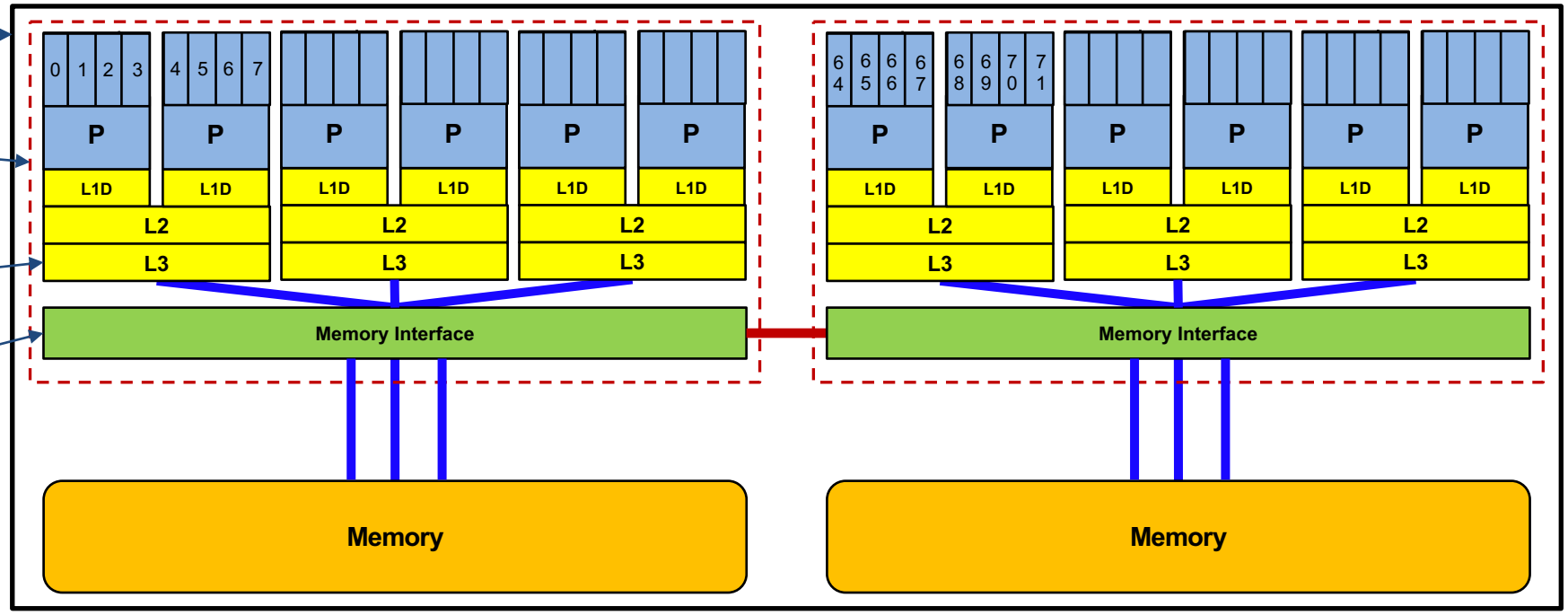
# likwid-topology

```
*************************************************************
GPU Topology
*************************************************************
GPU count:           4
-------------------------------------------------------------
ID:                  0
Name:                Tesla V100-SXM2-16GB
Compute capability:  7.0
L2 size:             6.00 MB
Memory:              16.00 GB
SIMD width:          32
Clock rate:          1530000 kHz
Memory clock rate:   877000 kHz
Attached to NUMA node:     -1
```

GPUs with own NUMA domain

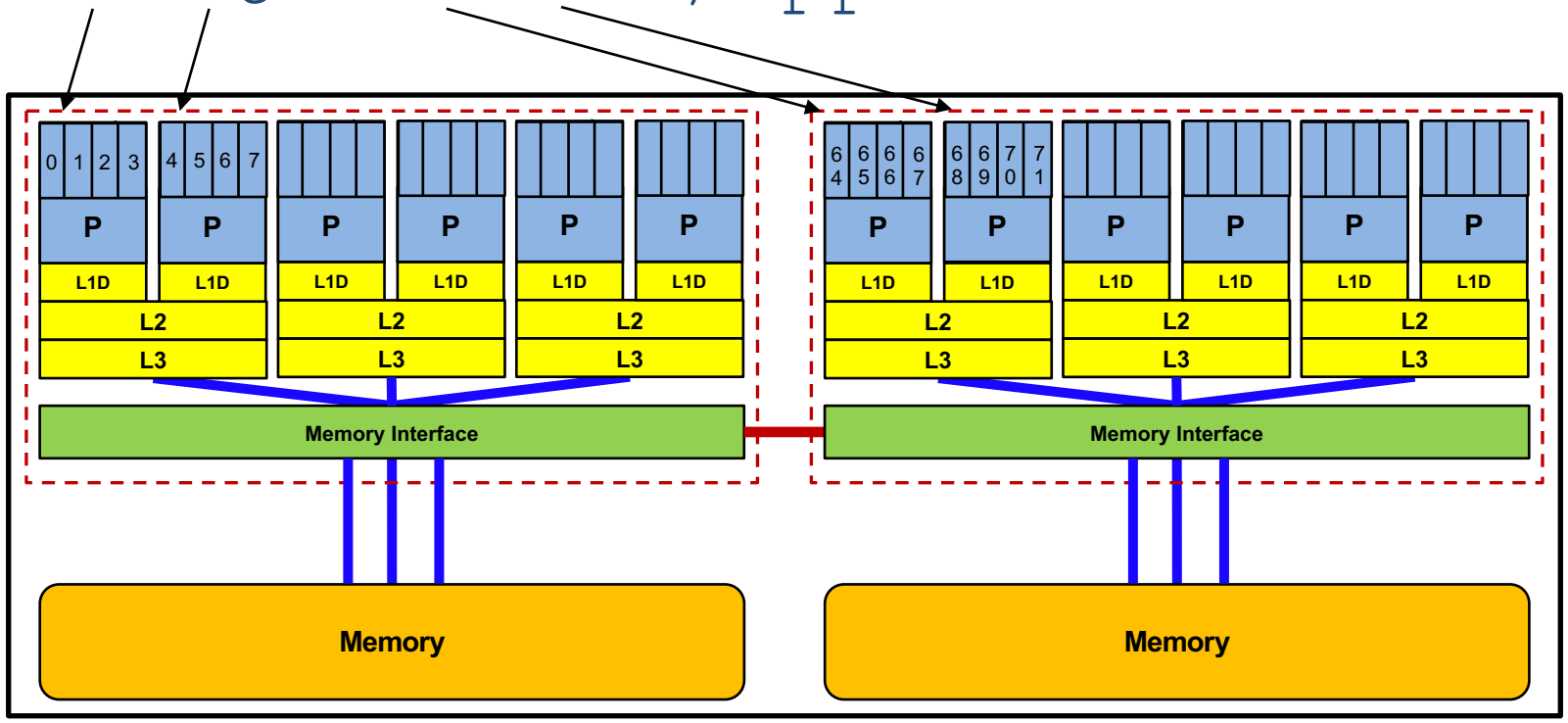
# likwid-pin

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



# likwid-pin

- `likwid-pin -c 0,1 ./app` == `taskset` with pinning
- `likwid-pin -c S0:0-1 ./app` `{0,4,8,…}`
  Domain S0, logical indicies 0,1 in phys. cores first sorted list
- `likwid-pin -c S0:0-1@S1:0-1 ./app`
  Combine selections



# likwid-perfctr

- Control performance measurements of x86_64, ARM, POWER and Nvidia GPUs (CUPTI)
- 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>`
  `-G <gpus> -W <gpuevents>`
  `(-m) (-t <time>) ./app`

GPU measurements only with MarkerAPI

With pinning

Activate MarkerAPIs

Timeline mode

# likwid-perfctr (MARCONI100)

- LIKWID available as module for workshop:
  ```
  module use /m100_work/tra20_TW36/tools/modulefiles
  module load likwid
  ```
- For GPU measurements: `module load cuda`
  Ensure CUPTI lib is in linker runtime path:
  ```
  export LD_LIBRARY_PATH=$CUDA_HOME/extras/CUPTI/lib64:$LD_LIBRARY_PATH
  ```
- All used codes in `/m100_work/tra20_TW36/likwid`

- No NEST counter measurements (perf_event_paranoid > 0) like memory traffic or NVlink
- No manipulation of frequencies/prefetchers

# likwid-perfctr (POWER9)

```
$ likwid-perfctr -C 0 -g L3 hostname
--------------------------------------------------------------------------------
CPU name:        POWER9, altivec supported
CPU type:        POWER9 architecture
CPU clock:       3.80 GHz
--------------------------------------------------------------------------------
r255n10
--------------------------------------------------------------------------------
Group 1: L3
+-------------------+---------+--------+
|       Event       | Counter | Core 0 |
+-------------------+---------+--------+
|   PM_L3_LD_PREF   |   PMC0  |   4780 |
|  PM_DATA_FROM_L3  |   PMC3  |    902 |
| PM_RUN_INST_CMPL  |   PMC4  | 254789 |
|    PM_RUN_CYC     |   PMC5  | 418966 |
+-------------------+---------+--------+
+-------------------------------+----------+
|             Metric            |  Core 0  |
+-------------------------------+----------+
|      Runtime (RDTSC) [s]      |   0.0021 |
|              CPI              |   1.6444 |
| L3D load bandwidth [MBytes/s] | 352.1851 |
| L3D load data volume [GBytes] |   0.0007 |
| Loads from local L3 per cycle |   1.3562 |
+-------------------------------+----------+
```

# Where to search 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?

# likwid-perfctr (POWER9)

```
#include <likwid-marker.h>
int main(int argc, char* argv[]) {
LIKWID_MARKER_INIT;
#pragma omp parallel {L
#pragma omp parallel {L
for (int i = 0; i < ite
    external_omp_call(s
}
#pragma omp parallel {L
LIKWID_MARKER_CLOSE;
}
```

- $CC -I$LIKWI
  -L$LIKWI
  -DLIKWID_
  … -llikwi

```
$ likwid-perfctr -C S0:0-1 -g FLOPS -m ./app
Region ompcall, Group 1: FLOPS
+--------------------+---------+----------+
|    Region Info     |  Core 0 |  Core 4  |
+--------------------+---------+----------+
| RDTSC Runtime [s] | 0.390438 | 0.390638 |
|     call count    |       10 |       10 |
+--------------------+---------+----------+
+-----------------------------------+----------+----------+
|               Metric              |  Core 0  |  Core 4  |
+-----------------------------------+----------+----------+
|        Runtime (RDTSC) [s]        |   0.3904 |   0.3906 |
|                CPI                |   1.8174 |   1.6775 |
| SP/DP [MFLOP/s] (scalar assumed)  | 1024.4920 | 1023.9662 |
|   SP [MFLOP/s] (vector assumed)   | 4097.9680 | 4095.8646 |
|   DP [MFLOP/s] (vector assumed)   | 2048.9840 | 2047.9323 |
+-----------------------------------+----------+----------+
```

# likwid-perfctr (Nvidia GPUs)

```
#include <likwid-marker.h>
int main(int argc, char* argv[]) {
LIKWID_NVMARKER_INIT;
LIKWID_NVMARKER_REGISTER("cudacall");
LIKWID_NVMARKER_START("cudacall");
for (int i = 0; i < iters; i++) {
    cudacall <double> <<<max_blocks, block_size>>>(dA, dB, dC, dD, buffer_size);
}
LIKWID_NVMARKER_STOP("cudacall");
LIKWID_NVMARKER_CLOSE;
}
```

- $CC -I$LIKWID_INCDI
  -L$LIKWID_LIBDI
  -DLIKWID_NVMON
  … -llikwid

```
$ likwid-perfctr -G 0 -W FLOPS_DP -m ./app
Region cudacall, Group 1: FLOPS_DP
+--------------------+----------+
|    Region Info     |   GPU 0  |
+--------------------+----------+
| RDTSC Runtime [s]  | 0.114596 |
|     call count     |       10 |
+--------------------+----------+
+--------------+------------+
|    Metric    |    GPU 0   |
+--------------+------------+
| DP [MFLOP/s] | 15142.8005 |
+--------------+------------+
```

# likwid-mpirun (POWER9)

- `$ likwid-mpirun -np 2 -t 10 ./app`
  Two MPI processes with 10 OMP threads each
- `$ likwid-mpirun -nperdomain M:1 -t 12 ./app`
  One MPI process per NUMA domain with 12 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
- MarkerAPI for GPUs supported

Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# CPU example

## Dense matrix-vector-multiplication

High Performance Computing

# Dense DP matrix-vector-multiplication

- Common operation in HPC codes

```
!$OMP PARALLEL DO PRIVATE(r)
do c = 1 , SIZE=1000
  tmp=x(c)
  do r = 1 , SIZE
    y(r)=y(r) + A(r,c)* tmp
  enddo
enddo
!$OMP END PARALLEL DO
```

Per inner loop:
- 1 ST
- 2 LD
- 2 FP ops

- 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

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

# 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 < N; i++)
    {
      for (j = 0; j < i; ++j) {
        cvec[i] += mat[offset+j]
                 * bvec[j];
      }
    }
  }
}
```

- 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 < N; i++)
    {
      for (j = 0; j < i; ++j) {
        cvec[i] += mat[offset+j]
                 * bvec[j];
      }
    }
  }
  LIKWID_MARKER_STOP("dMVM")
}
LIKWID_MARKER_CLOSE;
```

# Dense DP matrix-vector-multiplication

```
$ likwid-perfctr -C S0:0-2 -g <L2LOAD|L3> -m ./matrix
--------------------------------------------------------------------------------
CPU name:         POWER9, altivec supported
CPU type:         POWER9 architecture
CPU clock:        3.80 GHz
--------------------------------------------------------------------------------
Region dMVM, Group 1: L3
+----------------------+---------+-------------+-------------+-------------+
|         Event        | Counter |   Core 0    |   Core 4    |   Core 8    |
+----------------------+---------+-------------+-------------+-------------+
|    PM_RUN_INST_CMPL  |   PMC4  | 31268810000 | 18828330000 | 6328260000  |
|   L2 load data volume [GBytes]  |   28.9295   |    17.5066  |     4.3394  |
|   L3 load data volume [GBytes]  |    7.2360   |     4.6009  |     1.7578  |
+----------------------+---------+-------------+-------------+-------------+
```

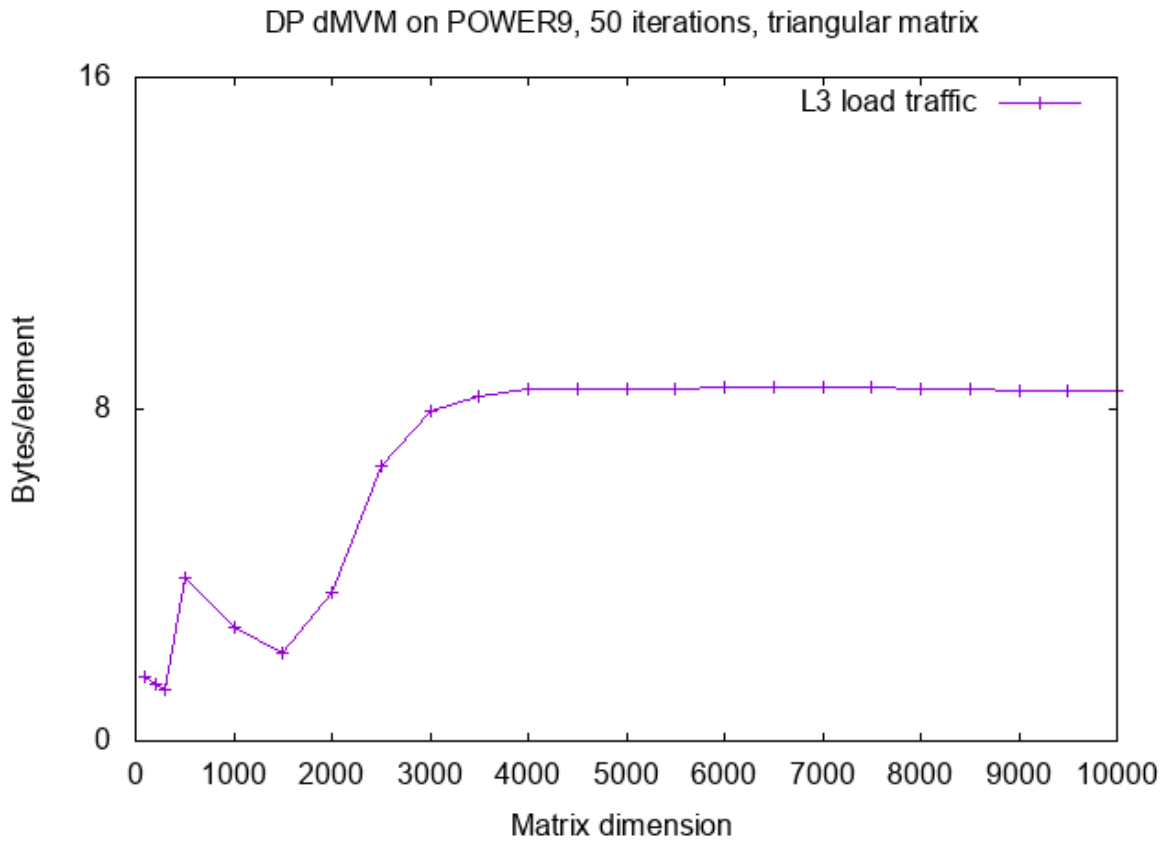
SIZE=8000
ROUNDS=50

First CPU loads most data into L1

# Dense DP matrix-vector-multiplication

- Load data traffic analysis:
  - Triangular matrix elements: $(N * (N + 1))/2 + N$
  - Iterations: 50
- For each element:
  - Only load matrix: 8 Byte
  - Load matrix and y: 16 Byte
  - Load matrix, y, and x: 24 Byte

Simple data traffic model seems to fit. But for matrix dims > 4000, y should come from L3 as well. Prefetchers kick in!

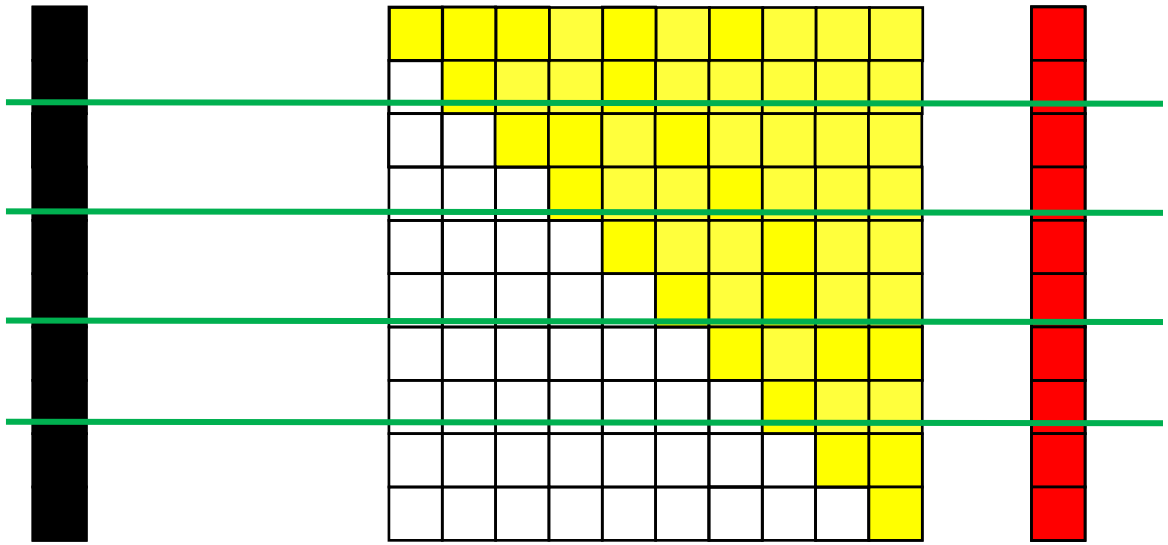


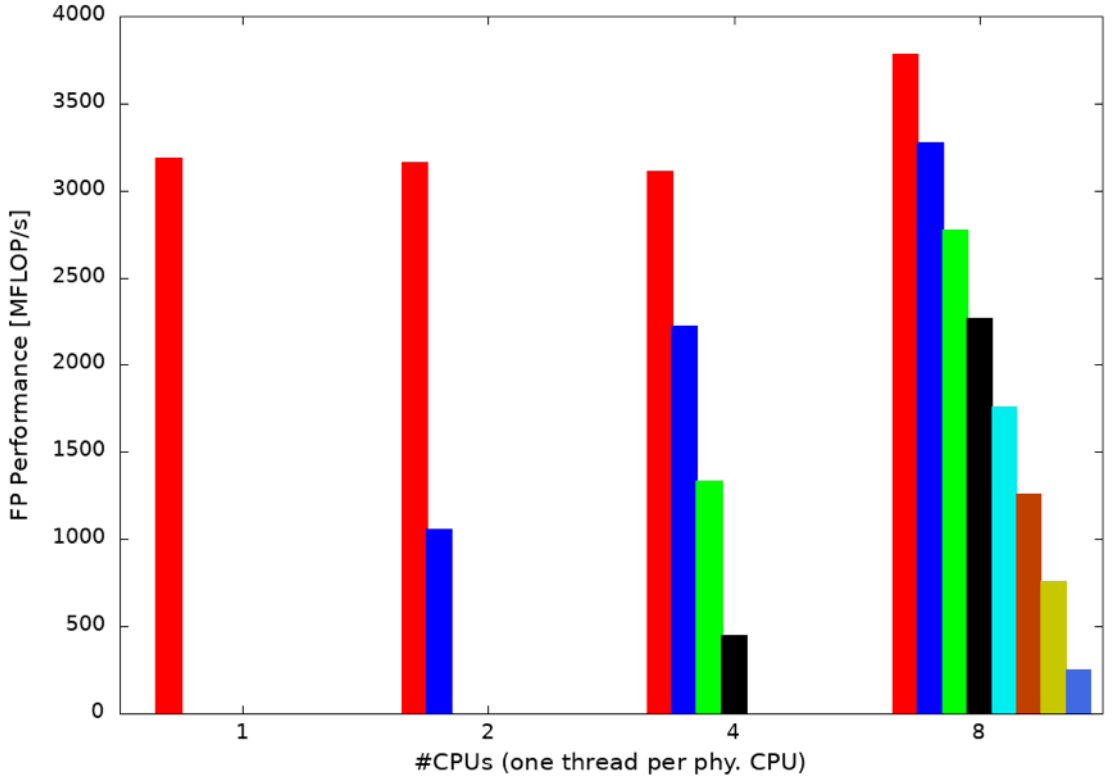
# Dense DP matrix-vector-multiplication

- 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)



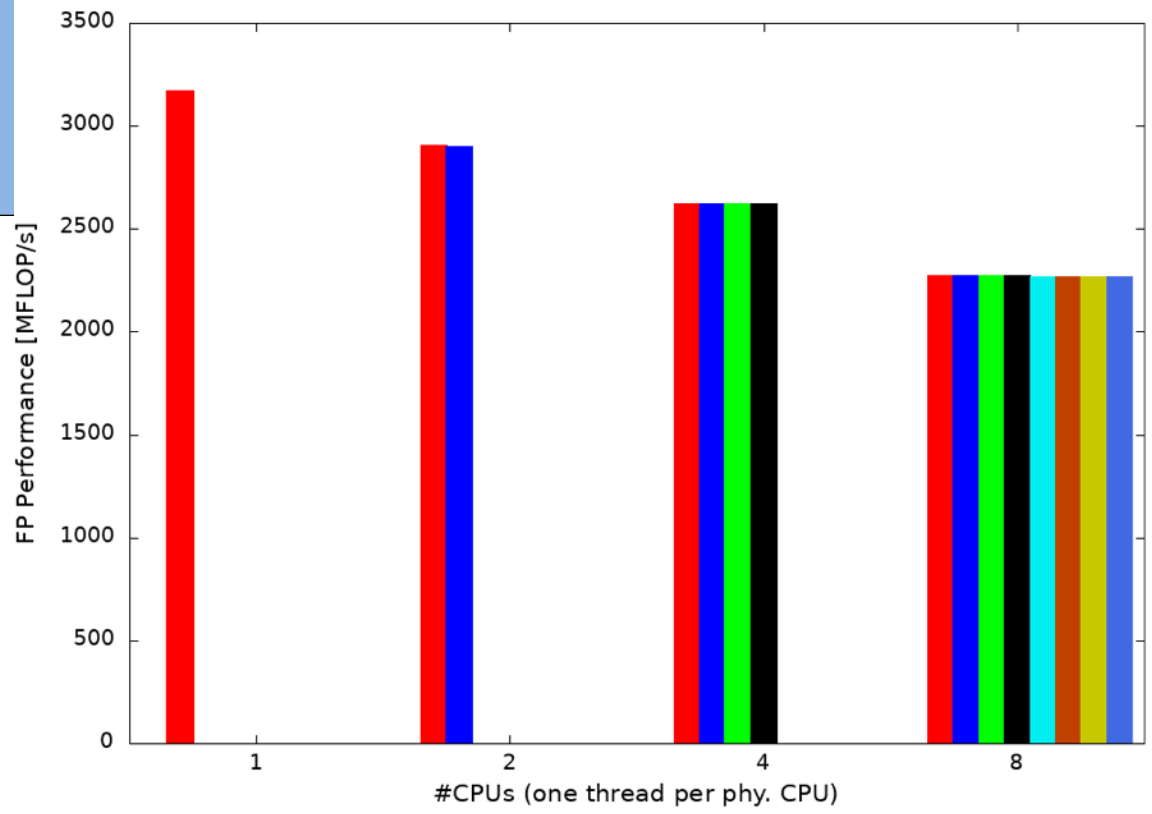


# Dense DP matrix-vector-multiplication

- 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)



Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# GPU example

2D DP Jacobi stencil (CUDA implementation)

# 2D DP Jacobi stencil (CUDA impl.)

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

- Stencil schemes frequently occur in PDE solvers on regular lattice structures
- Basically it is a sparse matrix vector multiply (spMVM) embedded in an iterative scheme (outer loop)
- But the regular access structure allows for matrix-free coding

```
do iter = 1, max_iterations
  Perform sweep over regular grid: y(:) ← x(:)
  Swap y ←→ x
enddo
```

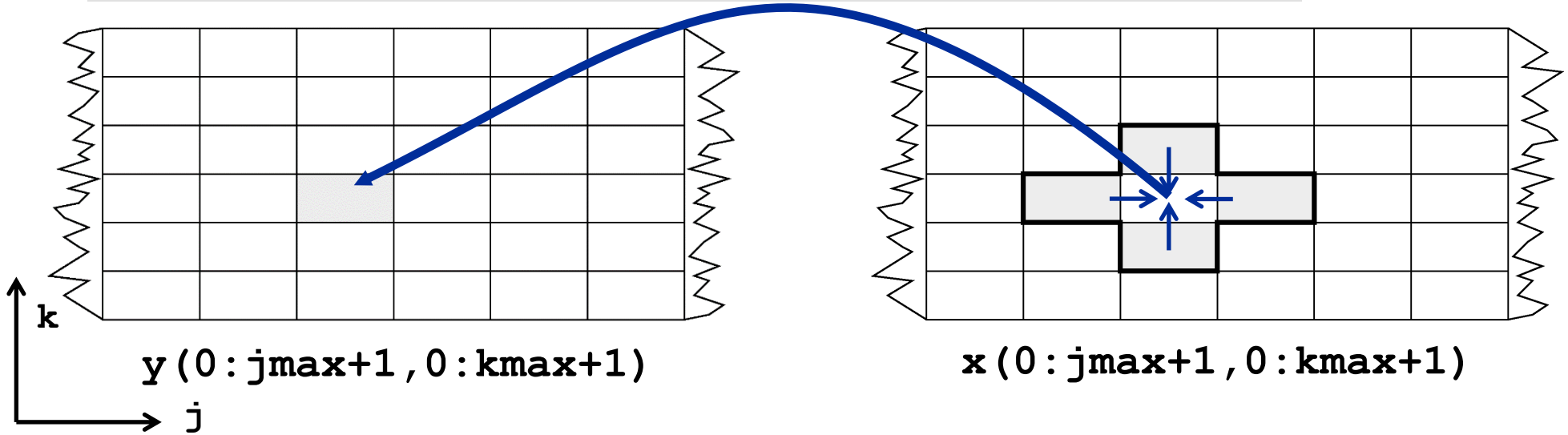
- Complexity of implementation and performance depends on
  - stencil operator, e.g. Jacobi-type, Gauss-Seidel-type, …
  - spatial extent, e.g. 7-pt or 25-pt in 3D,…

# 2D DP Jacobi stencil (CUDA impl.)

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
```

Lattice site update (LUP)

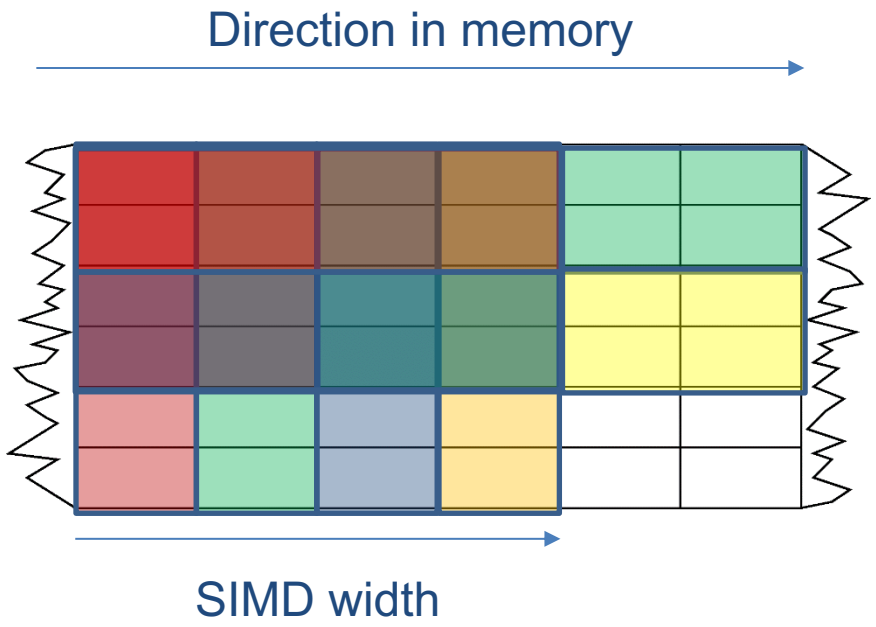



Appropriate performance metric: "**Lattice site updates per second**" [**LUP/s**]

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

# 2D DP Jacobi stencil (CUDA impl.)

- Work distribution to GPU threads is determined by the blocksize
- If cache lines need to be shared between threads, it causes performance degradation



# 2D DP Jacobi stencil (CUDA impl.)

- Let‘s examing two cases
  - `block(1, 256, 1)`
  - `block(64, 2, 1)`
- Grid splits up matrix according to block:
  `grid(width / block.x, height / block.y), 1);`
- Matrix size `4000x4000`

block(1, 256, 1)
Runtime: 1.22971 s
GLUP/s: 14.3124
Bandwidth: 228.99 GB/s

| Metric | GPU 0 |
|---|---|
| L1<-L2 global load bandwidth [MByte/s] | 401248.1169 |
| L1<-L2 global load data volume [GByte] | 433.8732 |

block(64, 2, 1)
Runtime: 0.477149 s
GLUP/s: 36.8858
Bandwidth: 590.17 GB/s

| Metric | GPU 0 |
|---|---|
| L1<-L2 global load bandwidth [MByte/s] | 400279.5772 |
| L1<-L2 global load data volume [GByte] | 134.4110 |

Less performance but higher data traffic at L2 due to ineffective cache line usage

Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# Summary

High Performance Computing

# Comments on LIKWID

- LIKWID for Nvidia Volta GPUs in beta state (https://github.com/RRZE-HPC/likwid/pull/325)
- GPU counters limited, many metrics not usable
- APIs partly under NDA → no documentation (no answers from contacts at Nvidia)

- LIKWID on POWER9 is stable
- Creation of performance groups difficult due to event-counter limitations
- Accuracy checks not complete yet!

# Summary

- LIKWID is a whole tool suite for running and profiling apps
- Provide easy-to-use tools for the daily work
- Supported for all relevant CPU architectures
- Nvidia GPU support in beta
- Main target: node-level performance engineering
- Try it: https://github.com/RRZE-HPC/likwid
  Thomas.Gruber@fau.de