

# PRACE Workshop: HPC code optimisation workshop Introduction to Likwid Thomas Gruber, Carla Guillen

LRZ | 2 – 4 November 2021

# Likwid – Like I knew what I am doing



- Lightweight command line tools for performance measurements.
- Open source. Software available at: <https://github.com/RRZE-HPC/likwid>

## Philosophy:

- Simple
- Efficient
- Portable
- Extensible

# Why Likwid?



Provides several functionalities:

- Performance profiling: Serial, MPI, OpenMP, Hybrid.
- Pinning
- Topology
- Energy Measurements
- Change processor frequency settings
- Provides a set of benchmarks (stream, daxpy, ddot, ...)

API to C, C++, Fortran, Python, Java

- There are several tools available for HPC applications.
- Profiling via advanced tools is often overkill
- A coarse overview is often sufficient

# The likwid tools (binaries)



Gather node architecture information:

- likwid-topology
- likwid-powermeter

Force affinity control and data placement:

- likwid-pin
- likwid-mpirun

Query and alter system settings:

- likwid-features
- likwid-setFrequencies

Performance profiling:

- likwid-perfctr
- Benchmarking:
- likwid-memsweeper
- likwid-bench

# Loading Likwid on CoolMUC3



- You can check available versions:  
`module av likwid`
- We will use the 5.2.0-gcc8 version  
`module load likwid/5.2.0-gcc8`

# Exploring likwid-perfctr



- Simple end-to-end measurement of hardware performance metrics
- Preconfigured and extensible metric groups, list with `likwid-perfctr -a`
- Operating modes:
  - Stethoscope
  - Timeline
  - Wrapper
  - Marker API

## Example of available groups:



BRANCH: Branch prediction miss rate/ratio

CLOCK: Clock of cores

DATA: Load to store ratio

FLOPS\_DP: Double Precision MFlops/s

FLOPS\_SP: Single Precision MFlops/s

L2: L2 cache bandwidth in MBytes/s

L2CACHE: L2 cache miss rate/ratio

L3: L3 cache bandwidth in MBytes/s

L3CACHE: L3 cache miss rate/ratio

MEM: Main memory bandwidth in MBytes/s

ENERGY: Energy consumption (RAPL)

## Stethoscope mode:



- likwid-perfctr measures on core base and has no notion what runs on the cores
- This enables to listen on what currently happens without any overhead:

```
likwid-perfctr -c N:0-11 -g FLOPS_DP -S 10s
```

- It can be used as cluster/server monitoring tool
- A frequent use is to measure a certain part of a long running parallel application from outside



## Timeline mode:



- Outputs the performance metrics in a specified frequency.

```
likwid-perfctr -C N:0-11 -g MEM -t 500ms ./a.out 2>  
out.txt
```

Wrapper mode:



Use `likwid-perfctr` as a wrapper of your serial or multi-threaded application.

```
likwid-perfctr -C S0:1 -g BRANCH ./a.out
```

Wrapper for MPI (only available with `mpi.intel`)

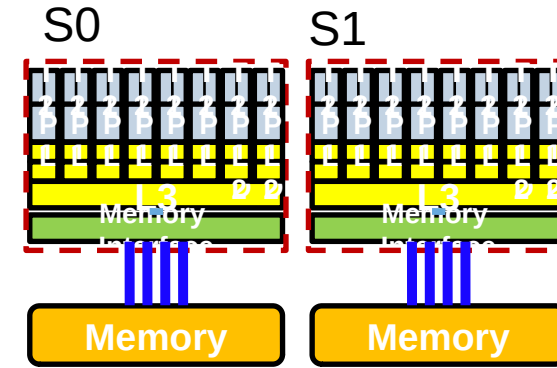
```
likwid-mpirun -np 32 -nperdomain S:8 -g CLOCK ./mympiexec
```

and Hybrid (only available with `mpi.intel`)

```
likwid-mpirun -np 32 -t 4 -g CLOCK ./mympiexec
```

```
{0,1,2,3}, {4,5,6,7}, {8,9,10,11}, ...
```

# Topology



`likwid-topology`

`likwid-pin -p`

**physical numbering:** processors are numbered according to the numbering in the OS

**logical numbering in node:** processors are logical numbered over whole node (N prefix)

**logical numbering in socket:** processors are logical numbered in every socket (S# prefix, e.g., S0)

**logical numbering in cache group:** processors are logical numbered in last level cache group (C# prefix, e.g., C1)

**logical numbering in memory domain:** processors are logical numbered in NUMA domain (M# prefix, e.g., M2)

**logical numbering within cpuset:** processors are logical numbered inside Linux cpuset (L prefix)

Example: `S0 : 0, 1`

## Marker API mode:



Modify your code by inserting calls to:

```
LIKWID_MARKER_INIT; //serial
```

```
LIKWID_MARKER_REGISTER("process"); //parallel
```

```
LIKWID_MARKER_START("process"); //parallel
```

```
[...]
```

```
LIKWID_MARKER_STOP("process"); //parallel
```

```
LIKWID_MARKER_CLOSE; //serial
```

Compile with `-DLIKWID_PERFMON` and execute with `likwid-perfctr -m ...`

# What to measure



- Likwid predefined performance groups help you.
- Performance groups compute derived metrics on a set of event.
- Examples:  
MEM, L3, L2, FLOPS\_DP, FLOPS\_SP, FLOPS\_AVX
- These groups have been validated by micro benchmarking.
- Performance measurements do not solve our performance problems. But: we get hints as to what problems we may have.

# Performance questions?



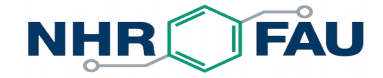
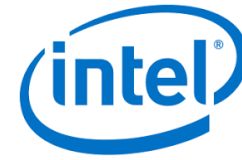
- ALU/Memory saturation?
- Load Imbalance?
- Vectorized floating point operations?
- Inefficient data access?
- Data between L1 and L2 Cache?
- Bad ccNUMA placement?
- Access between caches, memory and remote memory?
- Single or double precision? Flops/s ?
- Control flow problems?

```
for (int i=0;i<N;++i) {  
    array[i] *= array[i];  
}
```

## Demo on usage...

Download example from github:

```
> git clone https://github.com/carlabguillen/helloikwid.git
```



**Thank you!**



