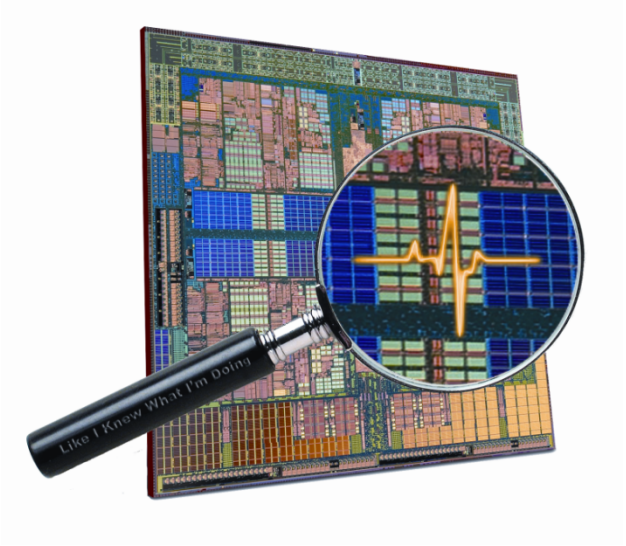


LIKWID

Lightweight performance tools

J. Treibig

Erlangen Regional Computing Center

University of Erlangen-Nuremberg

hpc@rrze.fau.de

BOF, ISC 2013
19.06.2013

Outline

- **Current state**
  - Overview
  - Building and installing likwid
  - likwid-topology and likwid-pin
  - likwid-powermeter
  - likwid-bench
  - likwid-perfctr
- **Outlook on next release**
  - New features
  - Current Problems
- **Plans and Ideas**

Likwid Tool Suite

- **Command line tools for Linux:**
  - easy to install
  - works with standard linux 2.6 kernel
  - simple and clear to use
  - supports Intel and AMD CPU

Open source project (GPL v2):

http://code.google.com/p/likwid/

J. Treibig, G. Hager, G. Wellein: ***LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments.*** Accepted for PSTI2010, Sep 13-16, 2010, San Diego, CA http://arxiv.org/abs/1004.4431

Why?

- **Question: There is tool XY? They can do the same thing. You are wasting your time.**

- **Possible answers:**
  - LIKWID has an unique feature set
  - LIKWID has NO external dependencies
  - LIKWID is easy to build and setup
  - LIKWID is just COOL (OK this is biased)

If you are still not convinced:

It is always good to have alternatives.

Even in Open Source tools.

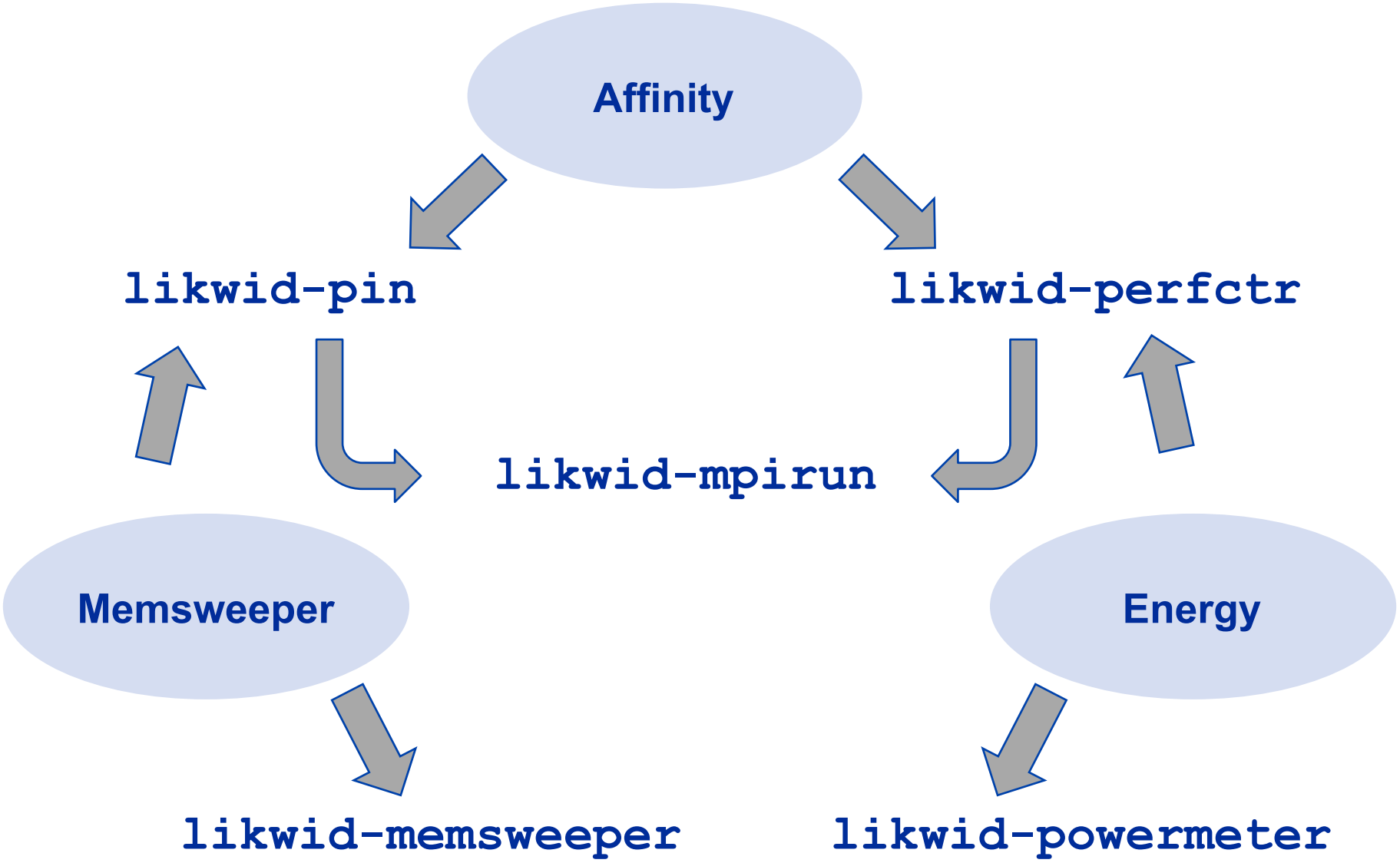
So try it and make your own opinion what suits your needs best.

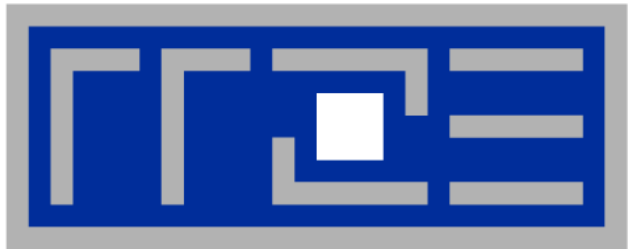
What is included in LIKWID?

Current release includes

- `likwid-topology` – Query node properties
- `likwid-pin` – Control affinity of serial and threaded programs
- `likwid-mpirun` – Control affinity of MPI and hybrid MPI/OpenMP programs
- `likwid-bench` – Microbenchmarking of node characteristics
- `likwid-memsweeper` – Clean up NUMA memory domains
- `likwid-powermeter` – Query Turbo mode steps and measure energy consumption on Intel SandyBridge systems
- `likwid-perfctr` – Measure Hardware Performance Monitoring data on X86 processors

Many functions in LIKWID are shared





Building LIKWID

Configuration

Options for access to hardware performance monitoring

Basics for building (for home use)

- Download the latest release from
  http://code.google.com/p/likwid/
- Read the INSTALL and README files ☺
- Also consider a look in the Wiki on the LIKWID website
- LIKWID has no external dependencies and should build on any Linux system with a 2.6 or newer kernel
- Installing is necessary for the pinning functionality and if you want to use the accessDaemon

Access to MSR and PCI Registers

- `likwid-perfctr` and `likwid-powermeter` require access to MSR (model-specific register) and (on SandyBridge) PCI registers.
- MSR registers are accessed on x86 processors via special instructions which can only be executed in kernel space
- The Linux kernel allows reading and writing to these registers via special device files.
- This enables to implement LIKWID completely in user space

The following options are available:

- Direct access to device files: The user must have read/write access to device files.
- AccessDaemon: The application starts a proxy application for access to device files (can be enabled in the Makefile).
- SysAccessDaemon: Central daemon with access control enabling usage of LIKWID as monitoring backend.

Setup direct access (for home use)

- All modern Linux distributions support the necessary msr kernel module

- Check if device file exists:

```
ls -l /dev/cpu/0/
```

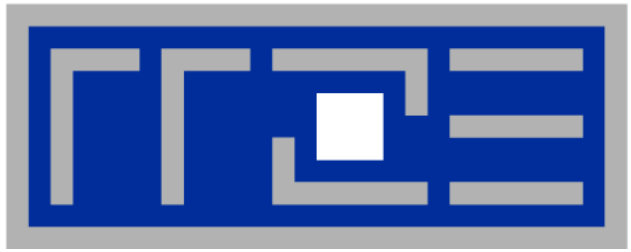
- If msr file is missing, load module (must be root):

```
modprobe msr
```

- Allow users access to msr device files (various solutions possible, must be root):

```
chmod o+rw /dev/cpu/*/msr
```

- Now you can use likwid-perfctr as normal user
- You can integrate the necessary steps in a startup script or configure udev



Scenario 1: Dealing with node properties and thread affinity

likwid-topology

likwid-powermeter

likwid-pin

likwid-topology

Single source of node information

- **Node information is usually scattered in various places**
- **likwid-topology provides all information in a single reliable source**
- **All information is based directly on cpuid**
- **Features:**
  - Thread topology
  - Cache topology
  - NUMA topology
  - Detailed cache parameters (-c command line switch)
  - Processor clock (measured)
  - ASCII art output (-g command line switch)

Output of `likwid-topology -g`

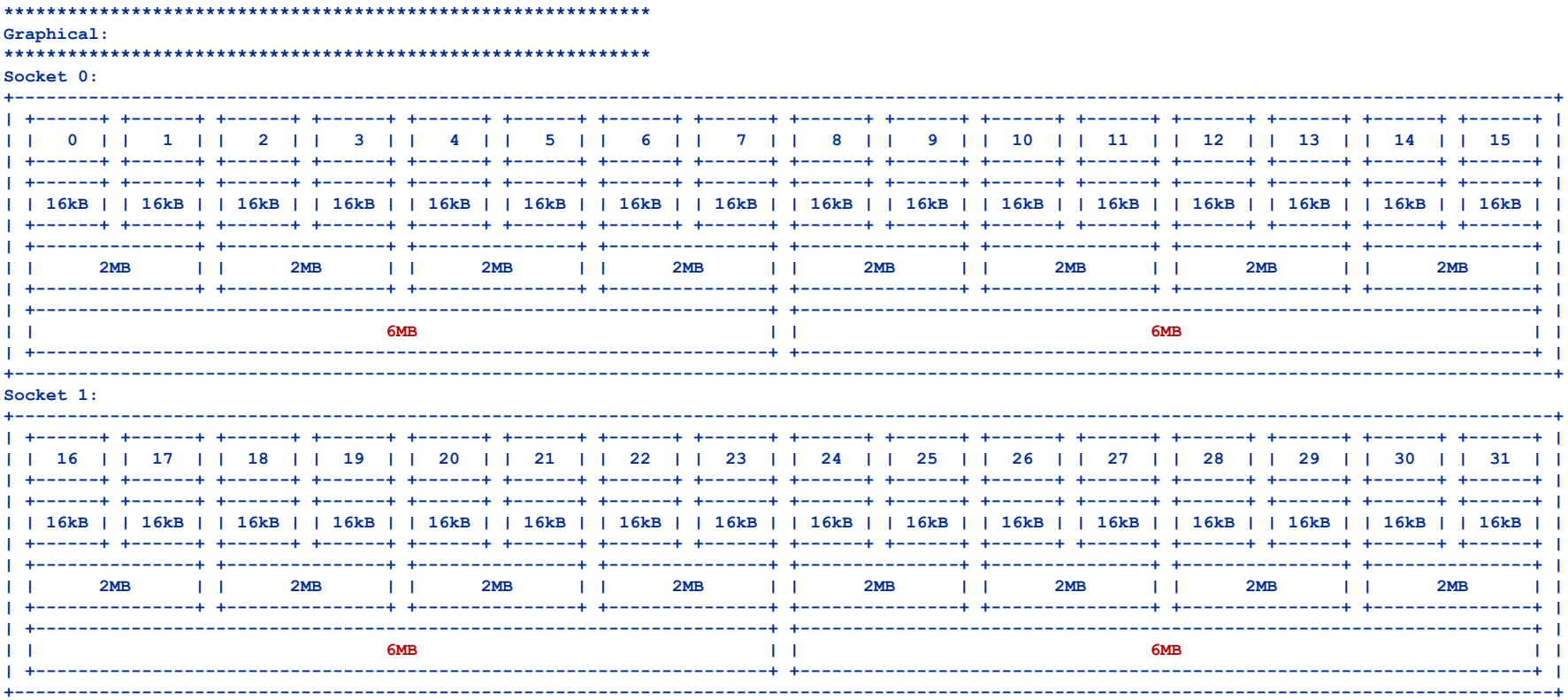
on one node of Cray XE6 "Hermit"

```
--------------------------------------------------------------------------------
CPU type:       AMD Interlagos processor
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                2
Cores per socket:       16
Threads per core:       1
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket
0               0               0               0
1               0               1               0
2               0               2               0
3               0               3               0
[...]
16              0               0               1
17              0               1               1
18              0               2               1
19              0               3               1
[...]
--------------------------------------------------------------------------------
Socket 0: ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
Socket 1: ( 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 )
--------------------------------------------------------------------------------

********************************************************************************
Cache Topology
********************************************************************************
Level:  1
Size:   16 kB
Cache groups:   ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 )
( 13 ) ( 14 ) ( 15 ) ( 16 ) ( 17 ) ( 18 ) ( 19 ) ( 20 ) ( 21 ) ( 22 ) ( 23 ) ( 24 ) ( 25 ) ( 26 )
( 27 ) ( 28 ) ( 29 ) ( 30 ) ( 31 )
```

Output of likwid-topology continued

```
--------------------------------------------------------------
Level:  2
Size:   2 MB
Cache groups:   ( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 ) ( 8 9 ) ( 10 11 ) ( 12 13 ) ( 14 15 ) ( 16 17 ) ( 18
19 ) ( 20 21 ) ( 22 23 ) ( 24 25 ) ( 26 27 ) ( 28 29 ) ( 30 31 )
--------------------------------------------------------------
Level:  3
Size:   6 MB
Cache groups:   ( 0 1 2 3 4 5 6 7 ) ( 8 9 10 11 12 13 14 15 ) ( 16 17 18 19 20 21 22 23 ) ( 24 25 26
27 28 29 30 31 )
--------------------------------------------------------------

**************************************************************
NUMA Topology
**************************************************************
NUMA domains: 4
--------------------------------------------------------------
Domain 0:
Processors:  0 1 2 3 4 5 6 7
Memory: 7837.25 MB free of total 8191.62 MB
--------------------------------------------------------------
Domain 1:
Processors:  8 9 10 11 12 13 14 15
Memory: 7860.02 MB free of total 8192 MB
--------------------------------------------------------------
Domain 2:
Processors:  16 17 18 19 20 21 22 23
Memory: 7847.39 MB free of total 8192 MB
--------------------------------------------------------------
Domain 3:
Processors:  24 25 26 27 28 29 30 31
Memory: 7785.02 MB free of total 8192 MB
--------------------------------------------------------------
```

Output of likwid-topology continued



Likwid-pin
Overview

- **Pins processes and threads to specific cores without touching code**
- **Directly supports pthreads, gcc OpenMP, Intel OpenMP**
- **Based on combination of wrapper tool together with overloaded pthread library → binary must be dynamically linked!**
- **Can also be used as a superior replacement for taskset**
- **Supports logical core numbering within a node and within an existing CPU set**
  - Useful for running inside CPU sets defined by someone else, e.g., the MPI start mechanism or a batch system

- **Usage examples:**
  - `likwid-pin -c 0,2,4-6  ./myApp parameters`
  - `likwid-pin -c S0:0-3 ./myApp parameters`

- **Running the STREAM benchmark with likwid-pin:**

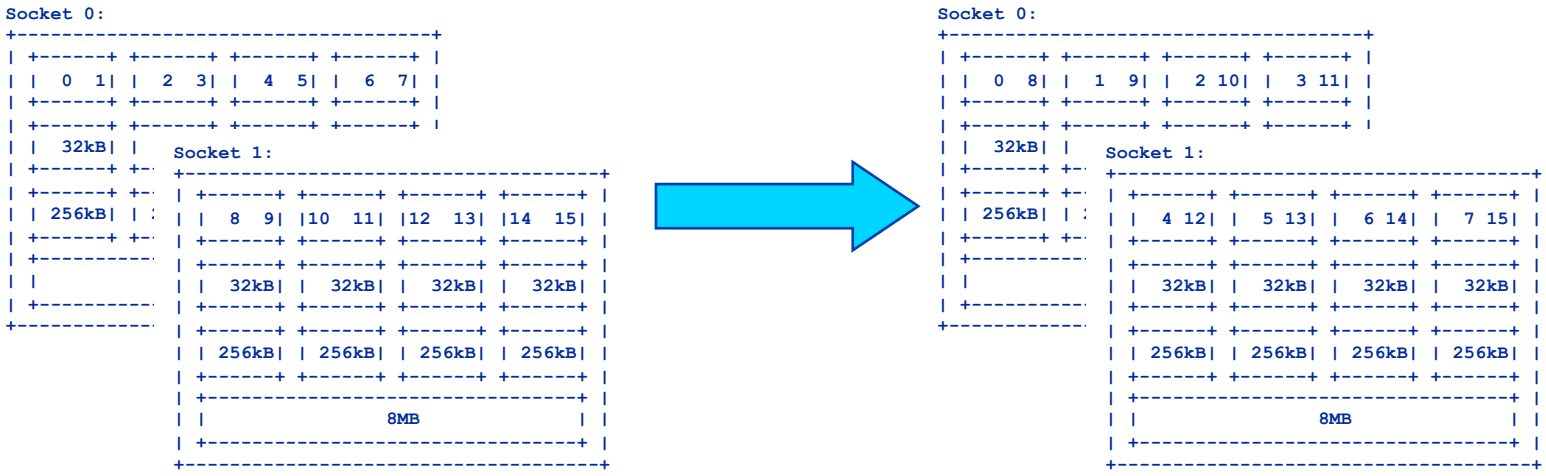
```
$ export OMP_NUM_THREADS=4
$ likwid-pin -c 0,1,4,5 ./stream
[likwid-pin] Main PID -> core 0 - OK
-----------------------------------------------
 Double precision appears to have 16 digits of accuracy
 Assuming 8 bytes per DOUBLE PRECISION word
-----------------------------------------------
[... some STREAM output omitted ...]
 The *best* time for each test is used
 *EXCLUDING* the first and last iterations
[pthread wrapper] PIN_MASK: 0->1  1->4  2->5
[pthread wrapper] SKIP MASK: 0x1
[pthread wrapper 0] Notice: Using libpthread.so.0
        threadid 1073809728 -> SKIP
[pthread wrapper 1] Notice: Using libpthread.so.0
        threadid 1078008128 -> core 1 - OK
[pthread wrapper 2] Notice: Using libpthread.so.0
        threadid 1082206528 -> core 4 - OK
[pthread wrapper 3] Notice: Using libpthread.so.0
        threadid 1086404928 -> core 5 - OK
[... rest of STREAM output omitted ...]
```

Main PID always pinned

Skip shepherd thread

Pin all spawned threads in turn

- **Core numbering may vary from system to system even with identical hardware**
  - Likwid-topology delivers this information, which can be fed into likwid-pin
- **Alternatively, likwid-pin can abstract this variation and provide a purely logical numbering in so called thread domains (physical cores first)**



- Across all cores in the node:

```
likwid-pin -c N:0-7 ./a.out
```

- Across the cores in each socket and across sockets in each node:

```
likwid-pin -c S0:0-3@S1:0-3 ./a.out
```

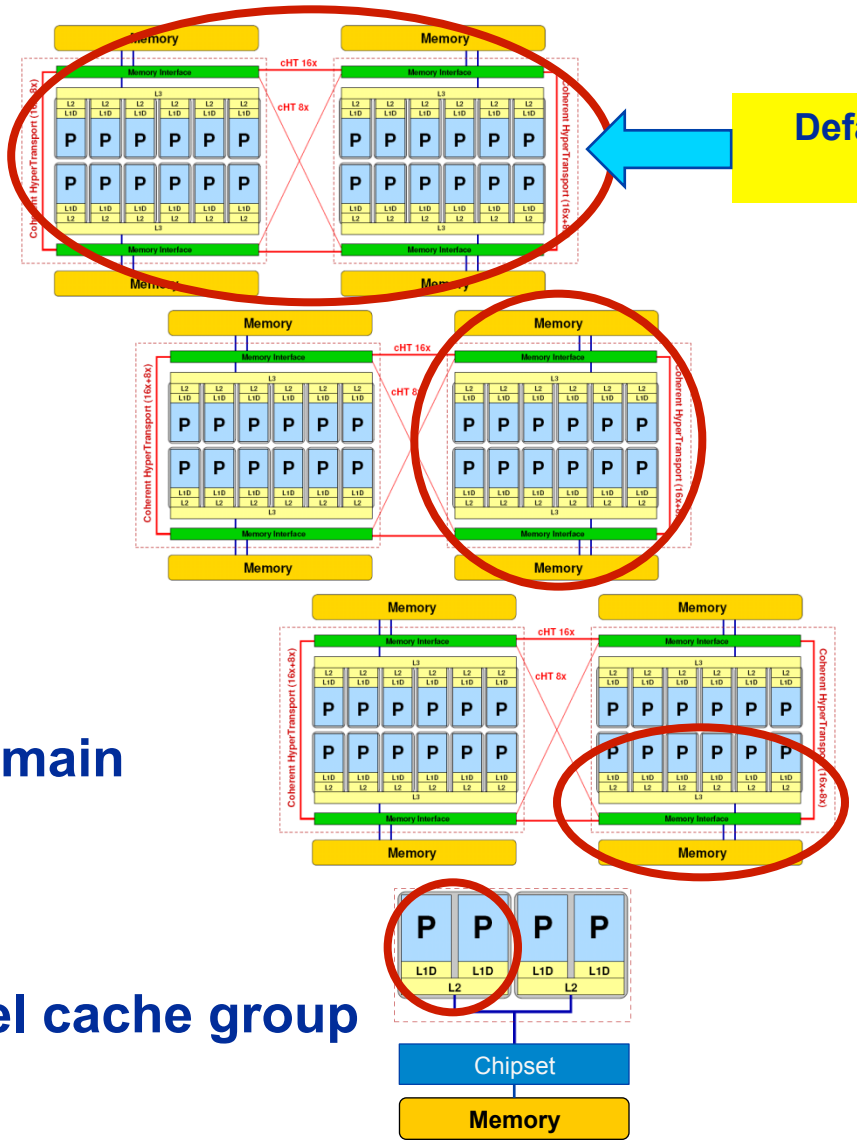
- **Possible unit prefixes**

N **node**

S **socket**

M **NUMA domain**

C **outer level cache group**



Default if –c is not specified!

Usage: likwid-pin

- Many options can be omitted, since `likwid-pin` has reasonable defaults

- `likwid-pin` will set `OMP_NUM_THREADS` for you with as many threads as you specify in your pinning expression

- With no options `likwid-pin` will use all processors:

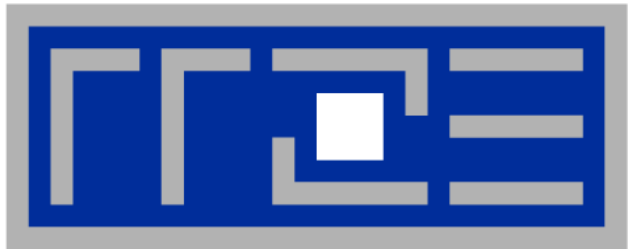
```
likwid-pin  ./stream-ICC
```

- `OMP_NUM_THREADS` will be set to 8:

```
likwid-pin -c S0:0-3@S1:0-3 ./stream-ICC
```

- `likwid-pin` can set NUMA page placement policy to interleaved using all NUMA domains used in your pinning expression:

```
likwid-pin -i -c S0:0-3@S1:0-3 ./stream-ICC
```



Scenario 2: Hardware performance monitoring and Node performance characteristics

likwid-bench

likwid-perfctr

- **Knowledge about the performance capabilities of a machine is essential for any optimization effort**

- **Microbenchmarking is an important method to gain this information**

`likwid-bench ...`

1. is an extensible, flexible benchmarking framework
2. allows rapid development of low level kernels
3. already includes many ready to use threaded benchmark kernels

- **Benchmarking runtime cares for:**
  - Thread management and placement
  - Data allocation and NUMA aware initialization
  - Timing and result presentation

likwid-bench Example

- Implement micro benchmark in abstract assembly
- The benchmark file is automatically converted, compiled and added to the benchmark application
- Benchmark files are located in the ./bench directory

```
$ likwid-bench -t clcopy -g 1 -i 1000 -w S0:1MB:2

$ likwid-bench -t load -g 2 -i 100 -w S1:1GB  -w S0:1GB-0:S1,1:S0
```

```
STREAMS 2
TYPE DOUBLE
FLOPS 0
BYTES 16
LOOP  32
movaps        FPR1, [STR0 + GPR1 * 8 ]
movaps        FPR2, [STR0 + GPR1 * 8 + 64 ]
movaps        FPR3, [STR0 + GPR1 * 8 + 128 ]
movaps        FPR4, [STR0 + GPR1 * 8 + 192 ]
movaps        [STR1 + GPR1 * 8 ], FPR1
movaps        [STR1 + GPR1 * 8 + 64 ], FPR2
movaps        [STR1 + GPR1 * 8 + 128 ], FPR3
movaps        [STR1 + GPR1 * 8 + 192 ], FPR4
```

Data streams used in benchmark

Flops performed and bytes transferred in one operation

Operations performed in one loop iteration

likwid-bench command line syntax

```
likwid-bench -h
likwid-bench -a
```
list available benchmarks

Required options:

```
likwid-bench -t copy -g 1 -w S1:1GB
-t <benchmark case>
-g <# thread groups>  need equivalent # working groups
-w <thread domain>:<working set size (kB, MB or GB)>
(-i <# iterations> adjust to get reasonable runtime)
```

Specify number of threads (Default: all processors in thread domain):

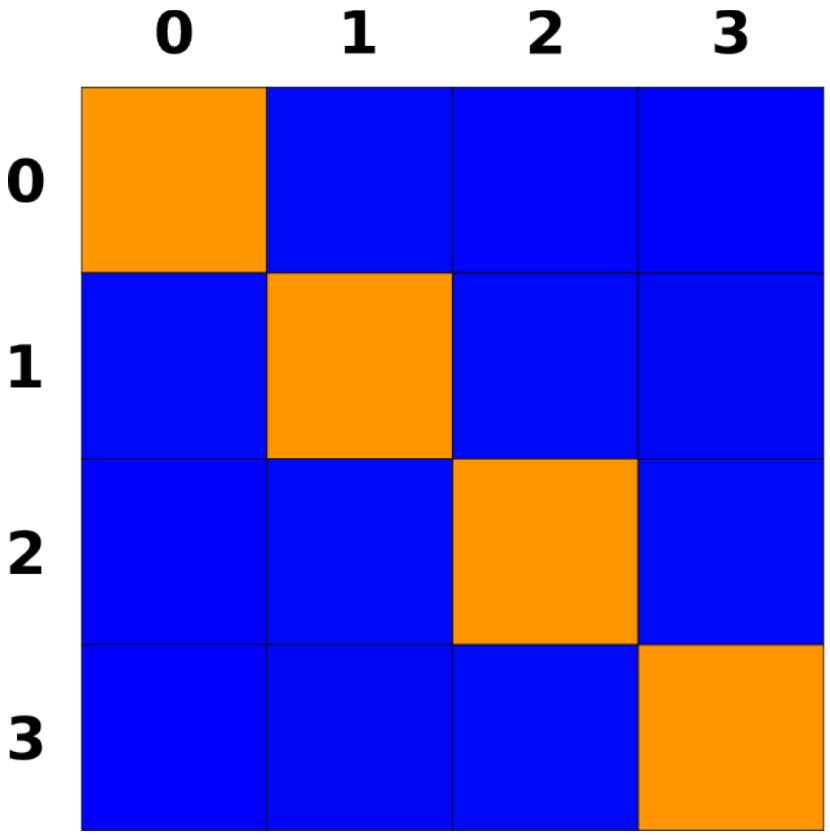
```
likwid-bench -t copy -g 1 -w S1:1GB:2
```

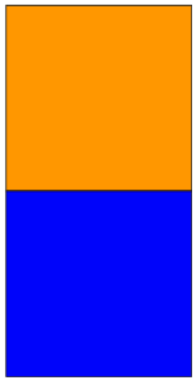
Specify data placement (Default: in same NUMA domain as threads):

```
likwid-bench -t copy -g 1 -w S1:1GB:2-0:S0,1:S1
```

Intel Nehalem EX 4-socket system

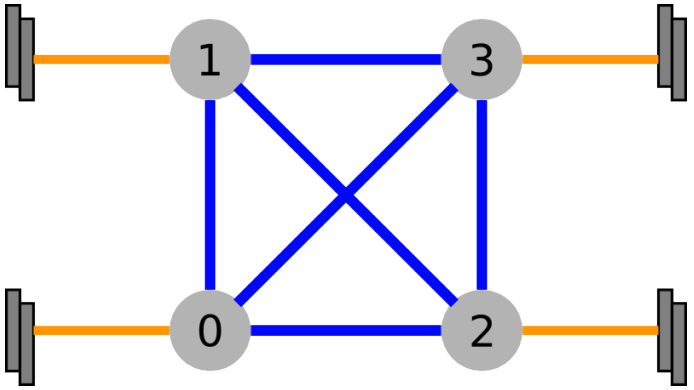
ccNUMA bandwidth map





15.5 GB/s

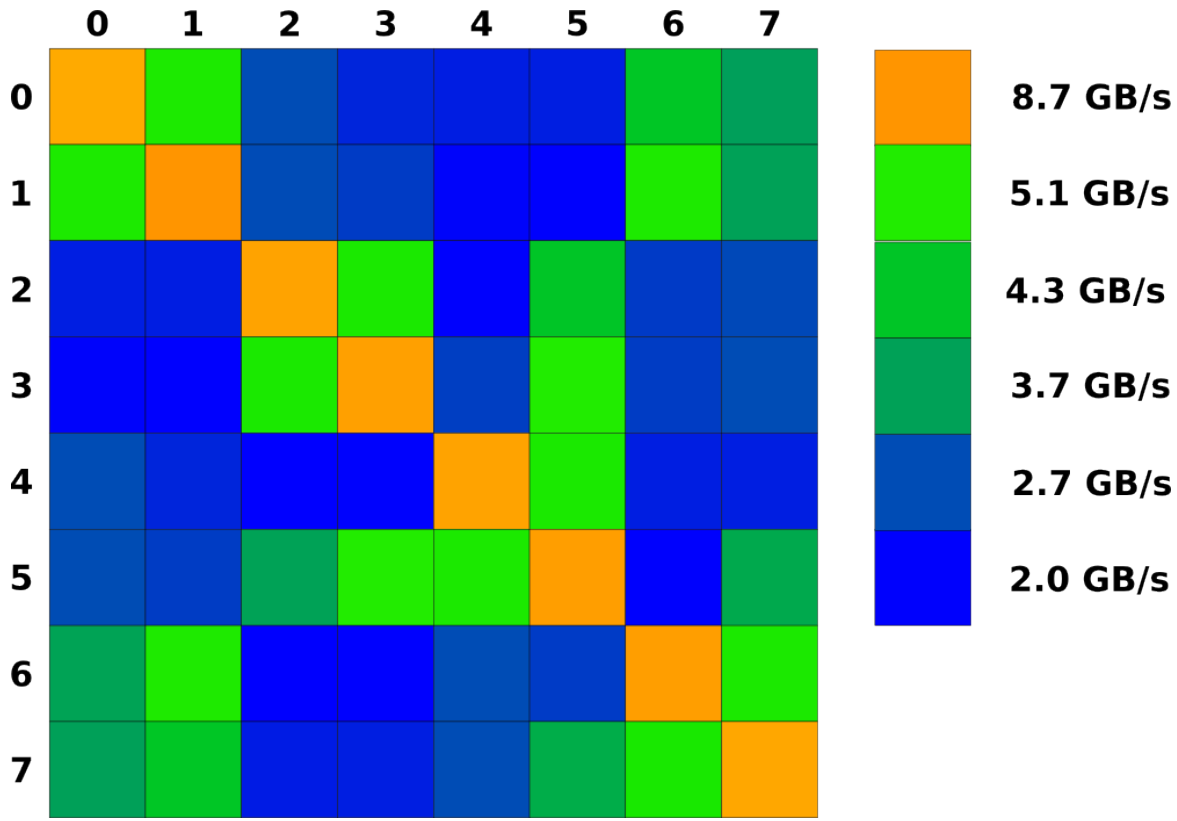
10.5 GB/s

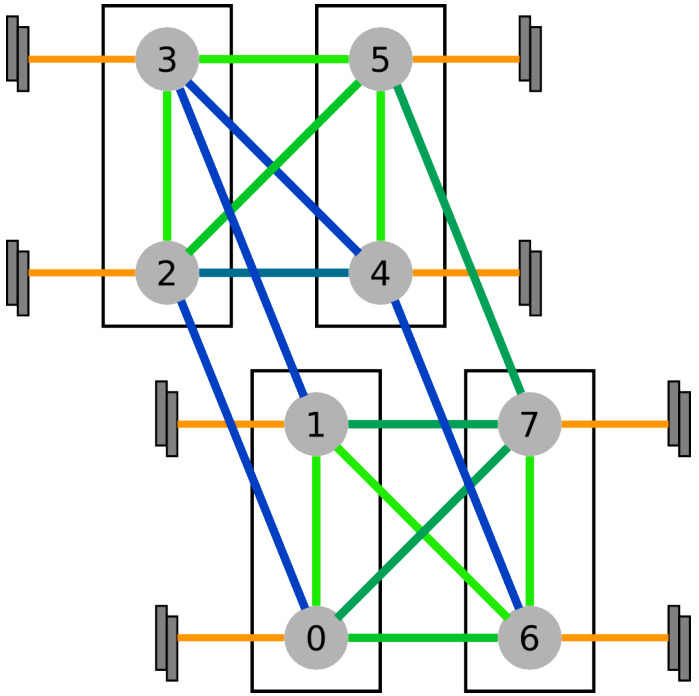


Bandwidth map data measured with likwid-bench. All cores used in one NUMA domain, memory is placed in a different NUMA domain. Test case: simple copy `A(:)=B(:)`, large arrays

AMD Magny Cours 4-socket system

Topology at its best?





- **A coarse overview of hardware performance monitoring data is often sufficient**
  - likwid-perfctr (similar to “perfex” on IRIX, “hpmcount” on AIX, “lipfpm” on Linux/Altix, “craypat” on Cray systems)
  - Simple end-to-end measurement of hardware performance metrics
  - Operating modes:
    1. Wrapper
    2. Stethoscope
    3. Timeline
    4. Marker API
  - Preconfigured and extensible metric groups, list with `likwid-perfctr -a`

```
BRANCH: Branch prediction miss rate/ratio
CACHE: Data cache miss rate/ratio
CLOCK: Clock of cores
DATA: Load to store ratio
FLOPS_DP: Double Precision MFlops/s
FLOPS_SP: Single Precision MFlops/s
FLOPS_X87: X87 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
TLB: TLB miss rate/ratio
```

Performance groups are preconfigured events sets connected to useful derived metrics!

1. **Runtime profile / Call graph (gprof)**
2. **Instrument those parts which consume a significant part of runtime**
3. **Find performance signatures**

Possible signatures:

- **Bandwidth saturation**
- **Instruction throughput limitation (real or language-induced)**
- **Latency impact (irregular data access, high branch ratio)**
- **Load imbalance**
- **ccNUMA issues (data access across ccNUMA domains)**
- **Pathologic cases (false cacheline sharing, expensive operations)**

likwid-perfctr

Example usage for Wrapper mode

```
$ likwid-perfctr -C N:0-3 -g FLOPS_DP   ./stream.exe
-------------------------------------------------------------
CPU type:       Intel Core Lynnfield processor
CPU clock:      2.93 GHz
-------------------------------------------------------------
Measuring group FLOPS_DP
-------------------------------------------------------------
YOUR PROGRAM OUTPUT
+-------------------------------------+------------+------------+------------+------------+
|                Event                |   core 0   |   core 1   |   core 2   |   core 3   |
+-------------------------------------+------------+------------+------------+------------+
|         INSTR_RETIRED_ANY           | 1.97463e+08 | 2.31001e+08 | 2.30963e+08 | 2.31885e+08 |
|       CPU_CLK_UNHALTED_CORE         | 9.56999e+08 | 9.58401e+08 | 9.58637e+08 | 9.57338e+08 |
|    FP_COMP_OPS_EXE_SSE_FP_PACKED    | 4.00294e+07 | 3.08927e+07 | 3.08866e+07 | 3.08904e+07 |
|    FP_COMP_OPS_EXE_SSE_FP_SCALAR    |     882     |      0      |      0      |      0      |
| FP_COMP_OPS_EXE_SSE_SINGLE_PRECISION |      0      |      0      |      0      |      0      |
| FP_COMP_OPS_EXE_SSE_DOUBLE_PRECISION | 4.00303e+07 | 3.08927e+07 | 3.08866e+07 | 3.08904e+07 |
+-------------------------------------+------------+------------+------------+------------+

+-------------------------+------------+---------+---------+---------+
|          Metric         |   core 0   | core 1  |  core 2 |  core 3 |
+-------------------------+------------+---------+---------+---------+
|       Runtime [s]       |  0.326242  | 0.32672 | 0.326801 | 0.326358 |
|           CPI           |  4.84647   | 4.14891 | 4.15061  | 4.12849  |
| DP MFlops/s (DP assumed) |  245.399   | 189.108 | 189.024  | 189.304  |
|      Packed MUOPS/s     |  122.698   | 94.554  | 94.5121  | 94.6519  |
|      Scalar MUOPS/s     | 0.00270351 |    0    |    0     |    0     |
|        SP MUOPS/s       |     0      |    0    |    0     |    0     |
|        DP MUOPS/s       |  122.701   | 94.554  | 94.5121  | 94.6519  |
+-------------------------+------------+---------+---------+---------+
```

Pinning build in

Always measured

Configured metrics (this group)

Derived metrics

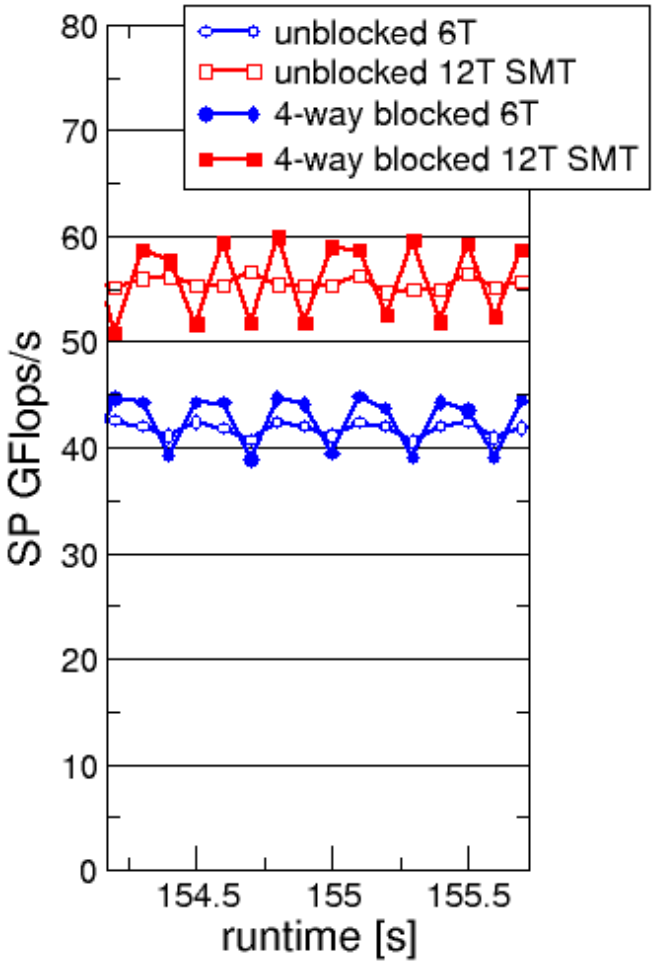
- likwid-perfctr measures on core base and has no notion what runs on the cores
- Use stethoscope mode to listen on what currently happens without any overhead:

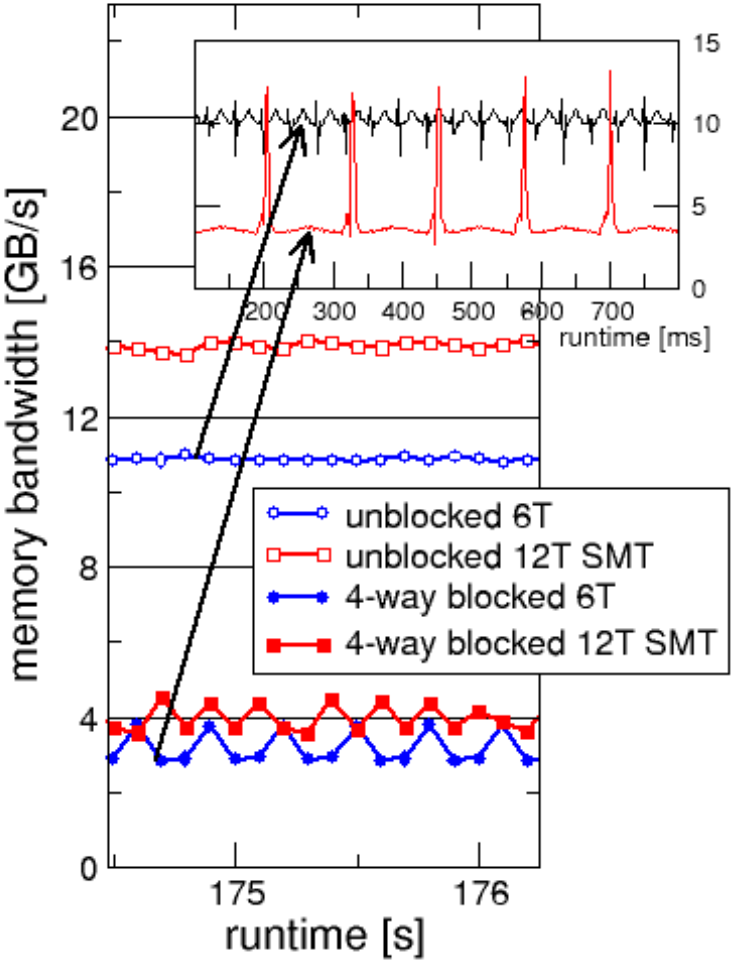
```
likwid-perfctr -c N:0-11 -g FLOPS_DP  -S 10
```

- 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

- **likwid-perfctr supports time resolved measurements of full node:**

```
likwid-perfctr -c N:0-11 -g MEM -t 50ms > out.txt
```





- To measure only parts of an application a marker API is available
- The API only turns counters on/off. The configuration of the counters is still done by the `likwid-perfctr` application
- Multiple named regions can be measured
- Results on multiple calls are accumulated
- Inclusive and overlapping regions are possible

```
#include <likwid.h>
likwid_markerInit();  // must be called from serial region

Likwid_markerThreadInit();  //Only if used in threaded setting
likwid_markerStartRegion("Compute");
. . .
likwid_markerStopRegion("Compute");


likwid_markerStartRegion("postprocess");
. . .
likwid_markerStopRegion("postprocess");

likwid_markerClose();  // must be called from serial region
```

- To enable easy toggling of instrumentation there is a set of macros

- To enable LIKWID instrumentation define `LIKWID_PERFMON`
- If `LIKWID_PERFMON` is undefined instrumentation will not be built

```
#define LIKWID_PERFMON // comment to disable
#include <likwid.h>

LIKWID_MARKER_INIT;

LIKWID_MARKER_THREADINIT;
LIKWID_MARKER_START("Compute");
. . .
LIKWID_MARKER_STOP("Compute");


LIKWID_MARKER_START("postprocess");
. . .
LIKWID_MARKER_STOP("postprocess");

LIKWID_MARKER_CLOSE;
```

likwid-perfctr
Group files

```
SHORT PSTI
EVENTSET
FIXC0 INSTR_RETIRED_ANY
FIXC1 CPU_CLK_UNHALTED_CORE
FIXC2 CPU_CLK_UNHALTED_REF
PMC0  FP_COMP_OPS_EXE_SSE_FP_PACKED
PMC1  FP_COMP_OPS_EXE_SSE_FP_SCALAR
PMC2  FP_COMP_OPS_EXE_SSE_SINGLE_PRECISION
PMC3  FP_COMP_OPS_EXE_SSE_DOUBLE_PRECISION
UPMC0  UNC_QMC_NORMAL_READS_ANY
UPMC1  UNC_QMC_WRITES_FULL_ANY
UPMC2 UNC_QHL_REQUESTS_REMOTE_READS
UPMC3 UNC_QHL_REQUESTS_LOCAL_READS
METRICS
Runtime [s] FIXC1*inverseClock
CPI  FIXC1/FIXC0
Clock [MHz]  1.E-06*(FIXC1/FIXC2)/inverseClock
DP MFlops/s (DP assumed) 1.0E-06*(PMC0*2.0+PMC1)/time
Packed MUOPS/s   1.0E-06*PMC0/time
Scalar MUOPS/s 1.0E-06*PMC1/time
SP MUOPS/s 1.0E-06*PMC2/time
DP MUOPS/s 1.0E-06*PMC3/time
Memory bandwidth [MBytes/s] 1.0E-06*(UPMC0+UPMC1)*64/time;
Remote Read BW [MBytes/s] 1.0E-06*(UPMC2)*64/time;
LONG
Formula:
DP MFlops/s =  (FP_COMP_OPS_EXE_SSE_FP_PACKED*2 +  FP_COMP_OPS_EXE_SSE_FP_SCALAR)/ runtime.
```

- Groups are architecture specific
- They are defined in simple text files
- The code is generated at compile time
- `likwid-perfctr -a` prints a list of available groups
- Information about a specific group is available with `-H -g <group>` switch

likwid-perfctr command line syntax

`likwid-perfctr -a` lists available performance groups

`likwid-perfctr -g <group> -H` performance group specific help

`likwid-perfctr -e [|less]` list available counters and raw events

Wrapper mode (with pinning):

```
likwid-perfctr -C N:0-3 -g L3  ./a.out
```

Wrapper mode (without pinning, application must pin itself):

```
likwid-perfctr -c N:0-3 -g L3  ./a.out
```

Stethoscope mode (duration in seconds):

```
likwid-perfctr -c S1:0-5 -g FLOPS_DP -S 4
```

Timeline mode (ms or s, output to stdout, must be further processed):

```
likwid-perfctr -c N:0-15 -g L2 -t 250ms [>out.txt]
```

Using instrumented binary (error if binary not instrumented):

```
likwid-perfctr -C N:0-3  -g L2 -m ./a.out
```

Remarks on likwid-perfctr

- `likwid-perfctr` performs simple start/stop measurements

- It does not know anything about the code running on the cores
- The connection between the measurement and your code is through pinning of processes and threads on the cores

This enables:

- Very accurate, low overhead measurements of even small code regions (using the Marker API)
- Usage as monitoring/profiling without the need to have access to the code or executable

Notice:

For an example on how timeline mode can be used to get a live monitoring tool have a look on `likwid-perfscope`.

Example: likwid-perfctr (1)

Identify load imbalance…

- `Instructions retired / CPI` may not be a good indication of useful workload – at least for numerical / FP intensive codes….
- **Floating Point Operations Executed** is often a better indicator
- Waiting / "Spinning" in barrier generates a high instruction count

Event	core 0	core 1	core 2	core 3	core 4	core 5
INSTR_RETIRED_ANY	2.10045e+10	1.90983e+10	1.729e+10	1.60898e+10	1.67958e+10	1.84689e+10
CPU_CLK_UNHALTED_CORE	1.82569e+10	1.81203e+10	1.81802e+10	1.82084e+10	1.82334e+10	1.82484e+10
CPU_CLK_UNHALTED_REF	1.66053e+10	1.6473e+10	1.65274e+10	1.65531e+10	1.65758e+10	1.65894e+10
FP_COMP_OPS_EXE_SSE_FP_PACKED	2.77016e+08	7.83476e+08	1.39355e+09	1.94365e+09	2.38059e+09	2.85981e+09
FP_COMP_OPS_EXE_SSE_FP_SCALAR	1.70802e+08	2.64065e+08	2.23153e+08	2.60835e+08	2.30434e+08	2.07293e+08
FP_COMP_OPS_EXE_SSE_SINGLE_PRECISION	19	0	0	0	0	0
FP_COMP_OPS_EXE_SSE_DOUBLE_PRECISION	4.47818e+08	1.04754e+09	1.61671e+09	2.20448e+09	2.61102e+09	3.0671e+09

Metric	core 0	core 1	core 2	core 3	core 4	core 5
Runtime [s]	6.84594	6.79471	6.81716	6.82773	6.83711	6.84274
Clock [MHz]	2932.07	2933.51	2933.51	2933.51	2933.51	2933.51
CPI	0.869191	0.948789	1.05148	1.13167	1.08559	0.988061
DP MFlops/s	109.192	275.833	453.48	624.893	751.96	892.857

```
!$OMP PARALLEL DO
DO I = 1, N
  DO J = 1, I
    x(I) = x(I) + A(J,I) * y(J)
  ENDDO
ENDDO
!$OMP END PARALLEL DO
```

Example: likwid-perfctr (2)

… and load-balanced codes

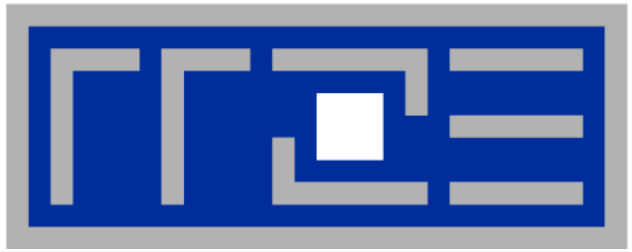
```
env OMP_NUM_THREADS=6 likwid-perfctr -t intel -C S0:0-5 -g FLOPS_DP ./a.out
```

Event	core 0	core 1	core 2	core 3	core 4	core 5
INSTR_RETIRED_ANY	1.83124e+10	1.74784e+10	1.68453e+10	1.66794e+10	1.76685e+10	1.91736e+10
CPU_CLK_UNHALTED_CORE	2.24797e+10	2.23789e+10	2.23802e+10	2.23808e+10	2.23799e+10	2.23805e+10
CPU_CLK_UNHALTED_REF	2.04416e+10	2.03445e+10	2.03456e+10	2.03462e+10	2.03453e+10	2.03459e+10
FP_COMP_OPS_EXE_SSE_FP_PACKED	3.45348e+09	3.43035e+09	3.37573e+09	3.39272e+09	3.26132e+09	3.2377e+09
FP_COMP_OPS_EXE_SSE_FP_SCALAR	2.93108e+07	3.06063e+07	2.9704e+07	2.96507e+07	2.41141e+07	2.37397e+07
FP_COMP_OPS_EXE_SSE_SINGLE_PRECISION	19	0	0	0	0	0
FP_COMP_OPS_EXE_SSE_DOUBLE_PRECISION	3.48279e+09	3.46096e+09	3.40543e+09	3.42237e+09	3.28543e+09	3.26144e+09

Metric	core 0	core 1	core 2	core 3	core 4	core 5
Runtime [s]	8.42938	8.39157	8.39206	8.3923	8.39193	8.39218
Clock [MHz]	2932.73	2933.5	2933.51	2933.51	2933.51	2933.51
CPI	1.22757	1.28037	1.32857	1.34182	1.26666	1.16726
DP MFlops/s	850.727	845.212	831.703	835.865	802.952	797.113
Packed MUOPS/s	423.566	420.729	414.03	416.114	399.997	397.101
Scalar MUOPS/s	3.59494	3.75383	3.64317	3.63663	2.95757	2.91165
SP MUOPS/s	2.33033e-06	0	0	0	0	0
DP MUOPS/s	427.161	424.483	417.673	419.751	402.955	400.013

Higher CPI but better performance

```
!$OMP PARALLEL DO
DO I = 1, N
  DO J = 1, N
    x(I) = x(I) + A(J,I) * y(J)
  ENDDO
ENDDO
!$OMP END PARALLEL DO
```



Measuring energy consumption with LIKWID

Measuring energy consumption

likwid-powermeter and likwid-perfctr -g ENERGY

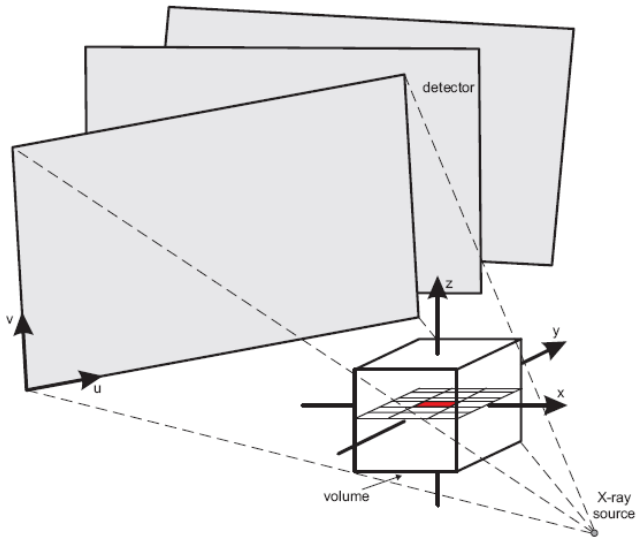
- **Implements Intel RAPL interface (Sandy Bridge)**
- **RAPL = “Running average power limit”**

```
-------------------------------------------------------------
CPU name:       Intel Core SandyBridge processor
CPU clock:      3.49 GHz
-------------------------------------------------------------
Base clock:     3500.00 MHz
Minimal clock:  1600.00 MHz
Turbo Boost Steps:
C1 3900.00 MHz
C2 3800.00 MHz
C3 3700.00 MHz
C4 3600.00 MHz
-------------------------------------------------------------
Thermal Spec Power: 95 Watts
Minimum  Power: 20 Watts
Maximum  Power: 95 Watts
Maximum  Time Window: 0.15625 micro sec
-------------------------------------------------------------
```

Example:

A medical image reconstruction code on Sandy Bridge

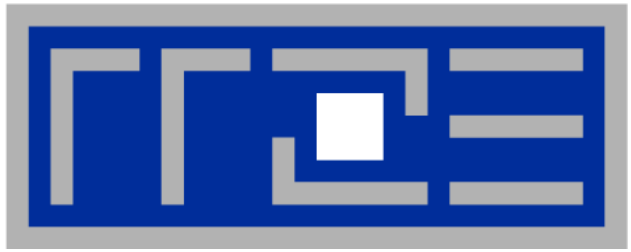
optional



Sandy Bridge EP (8 cores, 2.7 GHz base freq.)

Test case	Runtime [s]	Power [W]	Energy [J]
8 cores, plain C	**90.43**	90	8110
8 cores, SSE	29.63	93	2750
8 cores (SMT), SSE	22.61	102	2300
8 cores (SMT), AVX	**18.42**	111	2040

Faster code
less energy



Outlook on upcoming release

Overview of new features

- Support for Intel Haswell processor (Core events)
- Full Xeon Phi support (likwid-bench)
- likwid-pin can be used to convert logical to physical numberings
- New expression based syntax for processor lists in likwid-pin
- New workgroup syntax in likwid-bench (using compact placement)
- Many Bug Fixes (Fortran Marker API, Atom support, groups, NUMA thread groups, and many more)

New syntax in likwid-pin

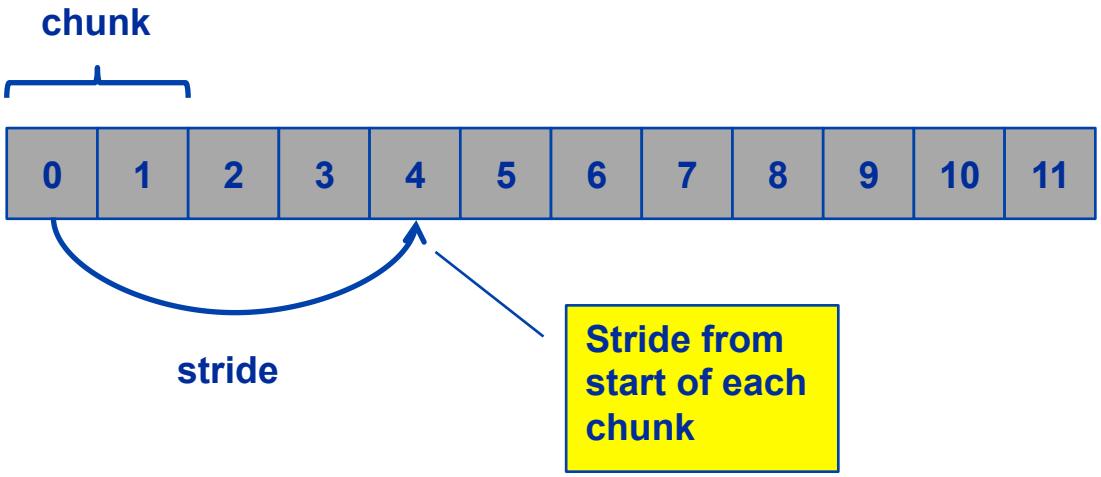
- **Default behaves the same as in old implementation:**

`likwid-pin -c S0:0-3` **using physical cores first**

- **New expression based syntax:**

`likwid-pin -c E:S0:4` **using compact ordering**

`likwid-pin -c E:S0:122:2:4` **with chunk size and stride**



New syntax in likwid-bench

- **Old syntax (using physical cores first ordering):**

`likwid-bench -w S0:1GB:4` **using processor 0, 1, 2, 3**

- **New syntax (using compact ordering):**

`likwid-bench -w S0:1GB:4` **using processor 0, 8, 1, 9**

- **New syntax options**

`likwid-bench -w S0:1GB:4:1:2` **using processor 0, 1, 2, 3**

Useful on systems with more than 2 SMT threads.

```
Socket 0:
+-------------------------------------+
| +------+ +------+ +------+ +------+ |
| |  0  8| |  1  9| |  2 10| |  3 11| |
| +------+ +------+ +------+ +------+ |
| +------+ +------+ +------+ +------+ |
| |  32kB| |  32kB| |  32kB| |  32kB| |
| +------+ +------+ +------+ +------+ |
| +------+ +------+ +------+ +------+ |
| | 256kB| | 256kB| | 256kB| | 256kB| |
| +------+ +------+ +------+ +------+ |
| +---------------------------------+ |
| |                8MB              | |
| +---------------------------------+ |
+-------------------------------------+
```

Known Problems

- likwid-mpirun needs much more work to be useful
- Missing documentation (Intel Xeon Phi Uncore, Haswell events)
- Multiplexing is missing (Xeon Phi has only 2 counters)
- PCI device based interface to Uncore on SandyBridge-EP is fragile and causes many problems
- New AMD architecture support is lagging behind (missing test machines)
- Kernel interface to MSR interface in the Linux kernel got more restrictive (no solution to this with NFS file systems)

Plans and New Ideas

Planned features:

- Multiplexing support
- More robust Uncore support on SandyBridge-EP
- Simple external locking mechanism for system monitoring
- Round robin option for likwid-pin

Plans:

- Measurement of overhead in LIKWID, PAPI and PERF
- Measurement of overhead for different counters/events
- Better validation of performance groups

Options:

- Alternative backend to perf kernel interface