Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Performance analysis with hardware metrics

likwid-perfctr

High Performance Computing

Probing performance behavior

- How do we find out about the performance properties and requirements of a parallel code?
  Profiling via advanced tools is often overkill
- A coarse overview is often sufficient: **`likwid-perfctr`**

- Simple end-to-end measurement of hardware performance metrics

Operating modes:

  - Wrapper
  - Stethoscope
  - Timeline
  - 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 frequency 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
ENERGY: Power and energy consumption
```

Best practices for Performance profiling

Focus on **resource utilization** and **instruction decomposition**!

Metrics to measure:

- Operation throughput (Flops/s)
- Overall instruction throughput (CPI)
- **Instruction breakdown**:
  - FP instructions
  - loads and stores
  - branch instructions
  - other instructions
- Instruction breakdown to **SIMD width** (scalar, SSE, AVX, AVX512 for X86). (only arithmetic instruction on most architectures)

- **Data volumes** and **bandwidths** to **main memory** (GB and GB/s)
- Data volumes and bandwidth to different **cache levels** (GB and GB/s)

Useful **diagnostic metrics** are:

- Clock frequency (GHz)
- Power (W)

All above metrics can be acquired using performance groups:

```
MEM_DP, MEM_SP, BRANCH, DATA, L2,  L3
```

likwid-perfctr wrapper mode

```
$ likwid-perfctr -g L2 –C S1:0-3 ./a.out
--------------------------------------------------------------------------------
CPU name:       Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz […]
--------------------------------------------------------------------------------
<<<< PROGRAM OUTPUT >>>>
--------------------------------------------------------------------------------
Group 1: L2
+-----------------------+---------+------------+------------+------------+------------+
|         Event         | Counter |  Core 14   |  Core 15   |  Core 16   |  Core 17   |
+-----------------------+---------+------------+------------+------------+------------+
|   INSTR_RETIRED_ANY   |  FIXC0  | 1298031144 | 1965945005 | 1854182290 | 1862521357 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  | 2353698512 | 2894134935 | 2894645261 | 2895023739 |
|  CPU_CLK_UNHALTED_REF |  FIXC2  | 2057044629 | 2534405765 | 2535218217 | 2535560434 |
|    L1D_REPLACEMENT    |   PMC0  |  212900444 |  200544877 |  200389272 |  200387671 |
|    L2_TRANS_L1D_WB    |   PMC1  |  112464863 |  99931184  |  99982371  |  99976697  |
|     ICACHE_MISSES     |   PMC2  |    21265   |    26233   |    12646   |    12363   |
+-----------------------+---------+------------+------------+------------+------------+
[… statistics output omitted …]
+---------------------------------+------------+------------+------------+------------+
|              Metric             |  Core 14   |  Core 15   |  Core 16   |  Core 17   |
+---------------------------------+------------+------------+------------+------------+
|       Runtime (RDTSC) [s]       |   1.1314   |   1.1314   |   1.1314   |   1.1314   |
|      Runtime unhalted [s]       |   1.0234   |   1.2583   |   1.2586   |   1.2587   |
|           Clock [MHz]           |  2631.6699 |  2626.4367 |  2626.0579 |  2626.0468 |
|               CPI               |   1.8133   |   1.4721   |   1.5611   |   1.5544   |
|  L2D load bandwidth [MBytes/s]  | 12042.7388 | 11343.8446 | 11335.0428 | 11334.9523 |
|  L2D load data volume [GBytes]  |   13.6256  |   12.8349  |   12.8249  |   12.8248  |
| L2D evict bandwidth [MBytes/s]  |  6361.5883 |  5652.6192 |  5655.5146 |  5655.1937 |
| L2D evict data volume [GBytes]  |   7.1978   |   6.3956   |   6.3989   |   6.3985   |
|    L2 bandwidth [MBytes/s]      | 18405.5299 | 16997.9477 | 16991.2728 | 16990.8453 |
|      L2 data volume [GBytes]    |   20.8247  |   19.2321  |   19.2246  |   19.2241  |
+---------------------------------+------------+------------+------------+------------+
```

Always measured for Intel CPUs

Configured metrics (this group)

Derived metrics

`likwid-perfctr` stethoscope mode

- likwid-perfctr counts events on cores; it has no notion of what kind of code is running (if any)

  This allows you to “listen” to what is currently happening, without any overhead:

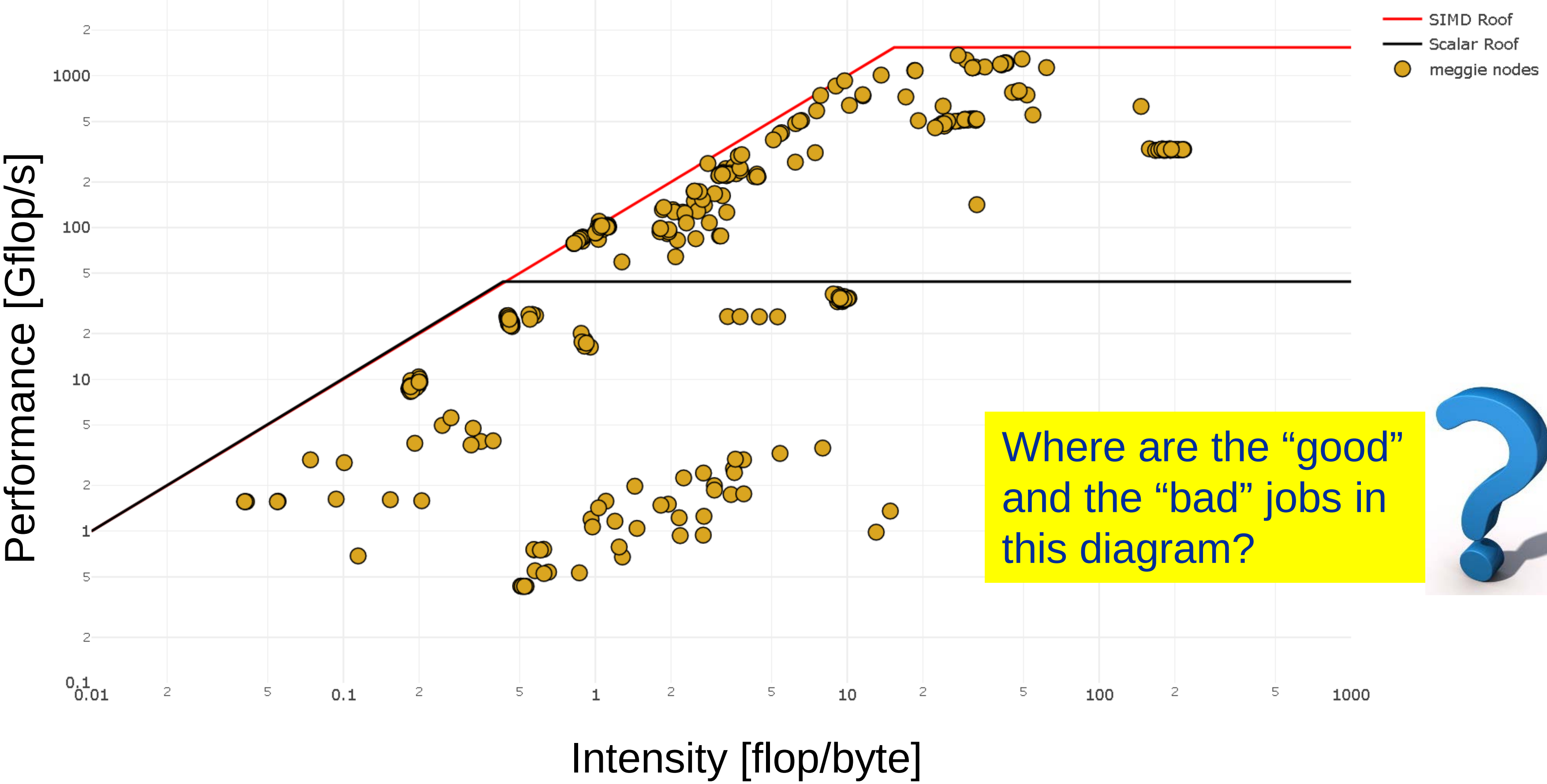
  ```
  $likwid-perfctr -c N:0-11 -g FLOPS_DP  -S 10s
  ```

- 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

Using Roofline for monitoring “live” jobs on a cluster

Based on measured BW and Flop/s data via likwid-perfctr



Where are the “good” and the “bad” jobs in this diagram?

likwid-perfctr marker API

- The marker API can restrict measurements to code regions
- The API only turns counters on/off. The configuration of the counters is still done by **likwid-perfctr**
- Multiple named regions support, accumulation over multiple calls
- Inclusive and overlapping regions allowed

```
#include <likwid-marker.h>
. . .
LIKWID_MARKER_INIT;               // must be called from serial region

. . .
LIKWID_MARKER_START(“Compute”);   // call markers for each thread
. . .
LIKWID_MARKER_STOP(“Compute”);
. . .
LIKWID_MARKER_START(“Postprocess”);
. . .
LIKWID_MARKER_STOP(“Postprocess”);
. . .


LIKWID_MARKER_CLOSE;              // must be called from serial region
```

Before LIKWID 5
use **likwid.h**

- Activate macros with **-DLIKWID_PERFMON**
- Run **likwid-perfctr** with **–m** switch to enable marking
- See **https://github.com/RRZE-HPC/likwid/wiki/TutorialMarkerF90** for Fortran example

Compiling, linking, and running with marker API

Compile:

```
cc -I /path/to/likwid.h -DLIKWID_PERFMON -c program.c
```

Link:

```
cc -L /path/to/liblikwid program.o -llikwid
```

Run:

```
likwid-perfctr -C <MASK> -g <GROUP> -m ./a.out
```

→ One separate block of output for every marked region

→ Caveat: Marker API can cause overhead; do not call too frequently!

Summary of hardware performance monitoring

- **Useful only if you know what you are looking for**
  - PM bears potential of acquiring massive amounts of data for nothing!
- **Resource-based metrics are most useful**
  - Cache lines transferred, work executed, loads/stores, cycles
  - Instructions, CPI, cache misses may be misleading

- **Caveat: Processor work != user work**
  - Waiting time in libraries (OpenMP, MPI) may cause lots of instructions
  - → distorted application characteristic

- **Another very useful application of PM: validating performance models!**
  - Roofline is data centric → measure data volume through memory hierarchy