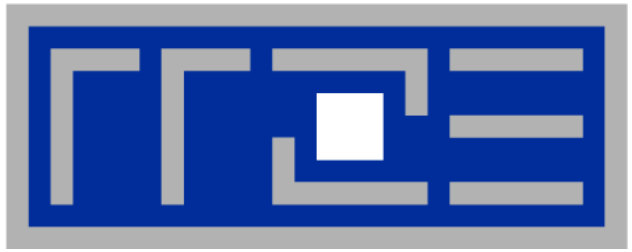


Performance patterns and hardware metrics on modern multicore processors: Best practices for performance engineering

Jan Treibig, Georg Hager, Gerhard Wellein
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg
Erlangen, Germany

PROPER Workshop at Euro-Par 2012
August 28, 2012

Rhodes Island, Greece

Hardware performance metrics

- … are ubiquitous as a starting point for performance analysis (including automatic analysis)
- … are supported by many tools
- … are often reduced to cache misses (what could be worse than cache misses?)

Reality:

- Modern parallel computing is plagued by bottlenecks
- There are typical performance patterns that cover a large part of possible performance behaviors
  - HPM signatures
  - Scaling behavior
  - Other sources of information

} "Performance pattern"

But still we need a HPM tool!

- **LIKWID: Lightweight command line tools for Linux**
- **Help to face the challenges without getting in the way**
- **Focus on x86 architecture**
- **Philosophy:**
  - Simple
  - Efficient
  - Portable
  - Extensible

Open source project (GPL v2):

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

Overview of LIKWID tools

- **Topology and Affinity:**
  - likwid-topology
  - likwid-pin
  - likwid-mpirun
- **Performance Profiling/Benchmarking:**
  - **likwid-perfctr**
  - likwid-bench
  - likwid-powermeter

Probing performance behavior with likwid-perfctr

- **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 (similar to “perfex” on IRIX, “hpmcount” on AIX, “lipfpm” on Linux/Altix)
  - 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 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
```

likwid-perfctr

Example usage with preconfigured metric group

```
$ env OMP_NUM_THREADS=4 likwid-perfctr -C N:0-3 -t intel -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  |
+--------------------------+------------+----------+----------+----------+
```

Always measured

Configured metrics (this group)

Derived metrics

- **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 likwid-perfctr application.**
- **Multiple named regions can be measured**
- **Results on multiple calls are accumulated**
- **Inclusive and overlapping Regions are allowed**

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

likwid_markerStartRegion("Compute");
. . .
likwid_markerStopRegion("Compute");


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

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

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
- **Code is generated on recompile**

Performance patterns (1)

Pattern	Peformance behavior	Metric signature
Load imbalance	Saturating/sub-linear speedup	Different amount of “work” on the cores (FLOPS_DP, FLOPS_SP, FLOPS_AVX); note that instruction count is not reliable!
BW saturation in outer-level cache	Saturating speedup across cores of OL cache group	OLC bandwidth meets BW of suitable streaming benchmark (L3)
Memory BW saturation	Saturating speedup across cores on a memory interface	Memory BW meets BW of suitable streaming benchmark (MEM)
Strided or erratic data access	Simple BW performance model much too optimistic	Low BW utilization / Low cache hit ratio, frequent CL evicts or replacements (CACHE, DATA, MEM)

Performance patterns (2)

Pattern	Peformance behavior	Metric signature
Bad instruction mix	Peformance insensitive to problem size vs. cache levels	Large ratio of instructions retired to FP instructions if the useful work is FP / Many cycles per instruction (CPI) if the problem is large-latency arithmetic / Scalar instructions dominating in data-parallel loops (FLOPS_DP, FLOPS_SP, CPI)
Limited instruction throughput	Large discrepancy from simple performance model based on LD/ST and arithmetic throughput	Low CPI near theoretical limit if instruction throughput is the problem / Static code analysis predicting large pressure on single execution port / High CPI due to bad pipelining (FLOPS_DP, FLOPS_SP, DATA)
Micro-architectural anomalies	Large discrepancy from performance model	Relevant events are very hardware-specific, e.g., stalls due to 4k memory aliasing, conflict misses, unaligned vs. aligned LD/ST, requeue events. Code review required, with architectural features in mind.

Performance patterns (3)

Pattern	Peformance behavior	Metric signature
Synchronization overhead	Speedup going down as more cores are added / No speedup with small problem sizes / Cores busy but low FP performance	Large non-FP instruction count (growing with number of cores used) / Low CPI (FLOPS_DP, FLOPS_DP, CPI)
False sharing of cache lines	Small speedup or slowdown when adding cores	Frequent (remote) CL evicts (CACHE)
Bad ccNUMA page placement	Bad or no scaling across NUMA domains	Unbalanced bandwidth on memory interfaces / High remote traffic (MEM)

The problem of instructions retired (1)

- 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
```

The problem of instructions retired (2)

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
```

Example 1:
Abstraction penalties in C++ code

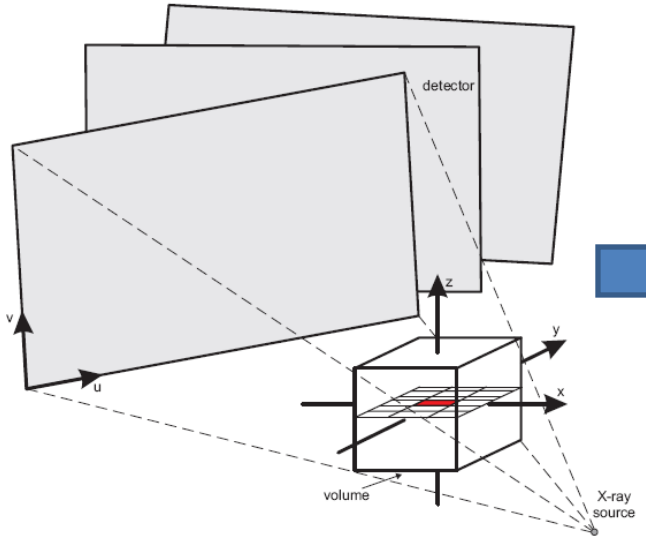
C++ codes which suffer from overhead (inlining problems, complex abstractions) need a lot more overall instructions related to the arithmetic instructions.

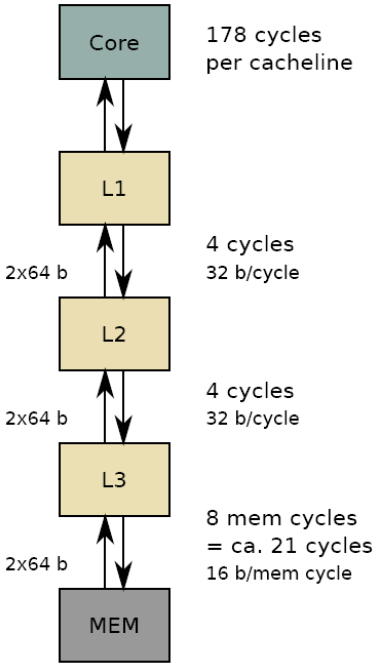
- Often (but not always) "good" (i.e., low) CPI → "Bad instruction mix" pattern
- Lower bandwidth
- Instruction throughput limited
- High-level optimizations complex or impossible → "Strided access" pattern

Example: Matrix-matrix multiply with expression template frameworks on a 2.93 GHz Westmere core

	Total retired instructions [10^{11}]	CPI	Memory Bandwidth [MB/s]	MFlops/s
Classic	**12.5**	**0.44**	**5300**	**1250**
Boost uBLAS	**10.1**	**4.6**	**630**	**156**
Eigen3	**2.1**	**0.41**	**371**	**8555**
Blaze/DGEMM	**2.0**	**0.32**	**531**	**11260**

Example 2:
Image reconstruction by backprojection





- **Simple roofline analysis**
  → Memory-bound algorithm → "Memory BW saturation" pattern
- **Closer look via `likwid-perfctr` MEM group and IACA tool**
  → "Limited instruction throughput" pattern
- **Work reduction optimization**
  → "Load imbalance" pattern identified by `likwid-perfctr` FLOPS_SP group → corrected by round-robin schedule

Conclusions

- **Automatic analysis is useful for the beginner, but will never match an experienced analyst**
- **Performance patterns are more than simple numbers**
  - Scaling behavior
  - Bottleneck saturation
  - HPM signatures
- **The set presented here is just a suggestion; it will have to be tested against more codes**
- **Power/energy patterns are still missing, but will have to be included**

Thank you.