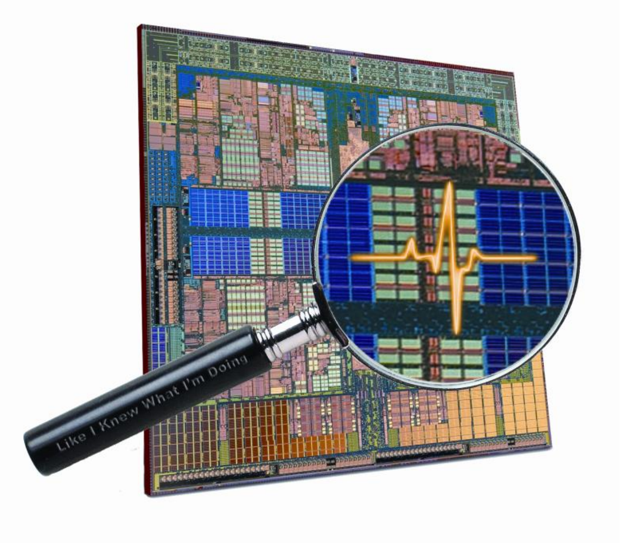


LIKWID – Lightweight Performance Tools

J. Treibig

HPC Services, Erlangen Regional Computing Center (RRZE)

9th VI-HPS Tuning Workshop

University Versailles

Likwid Lightweight Performance Tools

- **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

The following presentation will focus on `likwid-perfctr` and `likwid-powermeter`.

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 (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

likwid-perfctr
Stethoscope mode

- likwid-perfctr measures on core base and has no notion what runs on the cores

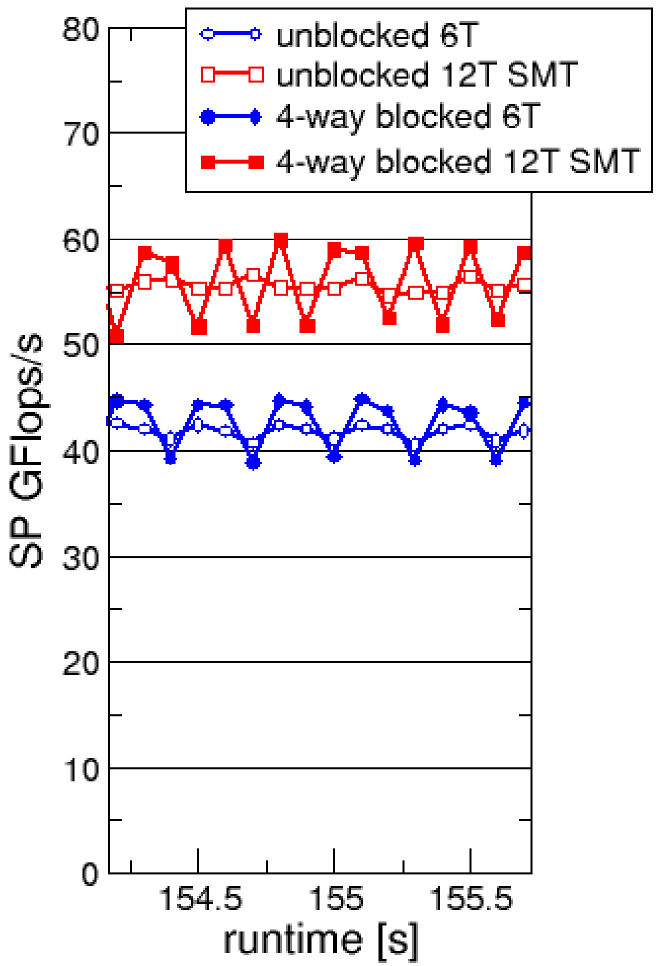
This enables to listen on what currently happens without any overhead:

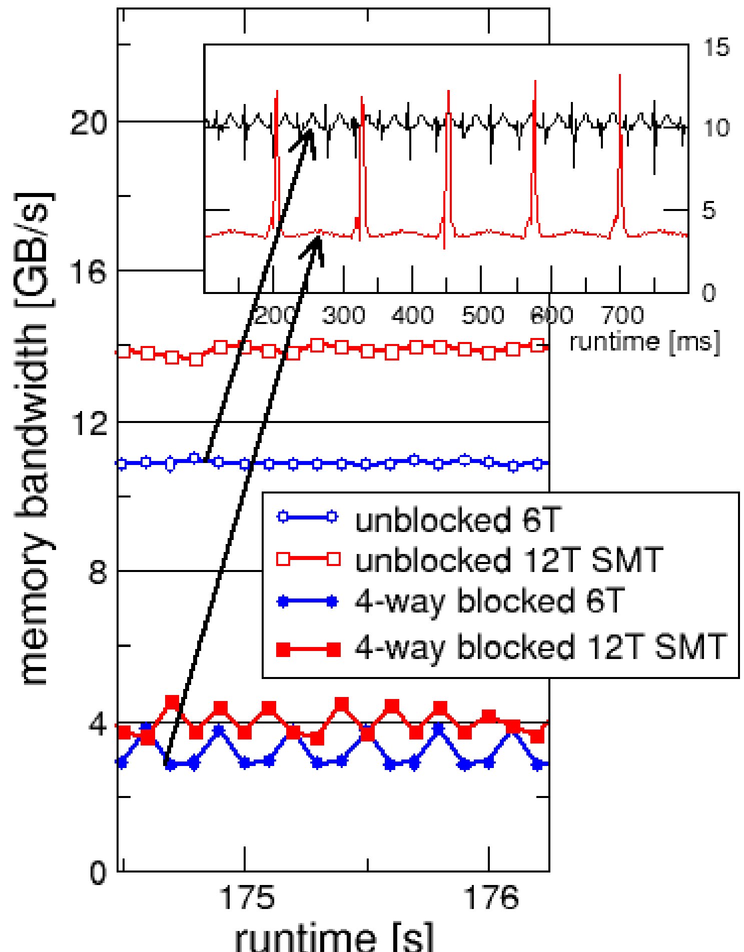
```
likwid-perfctr -c N:0-11 -g FLOPS_DP  -s 10
```

- 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 -d 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 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**
- **During recompile the code is generated**
- **likwid-perfctr -a outputs list of groups**
- **For every group an extensive documentation is available**

Things to look at (in roughly this order)

- Load balance (flops, instructions, BW)
- In-socket memory BW saturation, ccNUMA issues
- Shared cache BW saturation
- Flop/s, loads and stores per flop metrics
- SIMD vectorization
- CPI metric
- # of instructions, branches, mispredicted branches

Caveats

- Load imbalance may not show in CPI or # of instructions
  - Spin loops in OpenMP barriers/MPI blocking calls
  - Looking at "top" or the Windows Task Manager does not tell you anything useful
- In-socket performance saturation may have various reasons
- Cache miss metrics are overrated
  - If I really know my code, I can often *calculate* the misses
  - Runtime and resource utilization is much more important

likwid-perfctr

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
```

likwid-perfctr

… 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
```

Intel Nehalem EP node:

```
env OMP_NUM_THREADS=8 likwid-perfctr -g MEM -C N:0-7 \
                      -t intel ./a.out
```

Uncore events only counted once per socket

Event	core 0	core 1	[illegible]	[illegible]	core 4	core 5
INSTR_RETIRED_ANY	5.20725e+08	5.24793e+08	[illegible]	[illegible]	5.28269e+08	5.29083e+08
CPU_CLK_UNHALTED_CORE	1.90447e+09	1.90599e+09	1.90619e+09	1.90673e+09	1.90583e+09	1.90746e+09
UNC_QMC_NORMAL_READS_ANY	8.17606e+07	0	0	0	8.07797e+07	0
UNC_QMC_WRITES_FULL_ANY	5.53837e+07	0	0	0	5.51052e+07	0
UNC_QHL_REQUESTS_REMOTE_READS	6.84504e+07	0	0	0	6.8107e+07	0
UNC_QHL_REQUESTS_LOCAL_READS	6.82751e+07	0	0	0	6.76274e+07	0

RDTSC timing: 0.827196 s

Metric	core 0	core 1	core 2	core 3	core 4	core 5	core 6	core 7
Runtime [s]	0.714167	0.714733	0.71481	0.715013	0.714673	0.715286	0.71486	0.71515
CPI	3.65735	3.63188	3.65488	3.64076	3.60768	3.60521	3.59613	3.60184
Memory bandwidth [MBytes/s]	10610.8	0	0	0	10513.4	0	0	0
Remote Read BW [MBytes/s]	5296	0	0	0	5269.43	0	0	0

Half of read BW comes from other socket!

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

Example linear algebra with expression template frameworks:

	Total retired instructions [10^8]	Total arithmetic operations [10^7]	CPI	Memory Bandwidth [MB/s]	MFlops/s
Blitz++	2.84	5.13	**0.41**		
Blaze	0.46	5.05	1.12		

Identification of C++ overhead bound codes …

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

Example linear algebra with expression template frameworks:

	Total retired instructions [10^8]	Total arithmetic operations [10^7]	CPI	Memory Bandwidth [MB/s]	MFlops/s
Blitz++	**2.84**	5.13	**0.41**	4999	420
Blaze	0.46	5.05	1.12	10564	2909

- **Often very good CPI**
- **Lower bandwidth**
- **Overall instruction throughput limited**

Measuring energy consumption
likwid-powermeter

- **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
-------------------------------------------------------------
```

Energy consumption:
Example

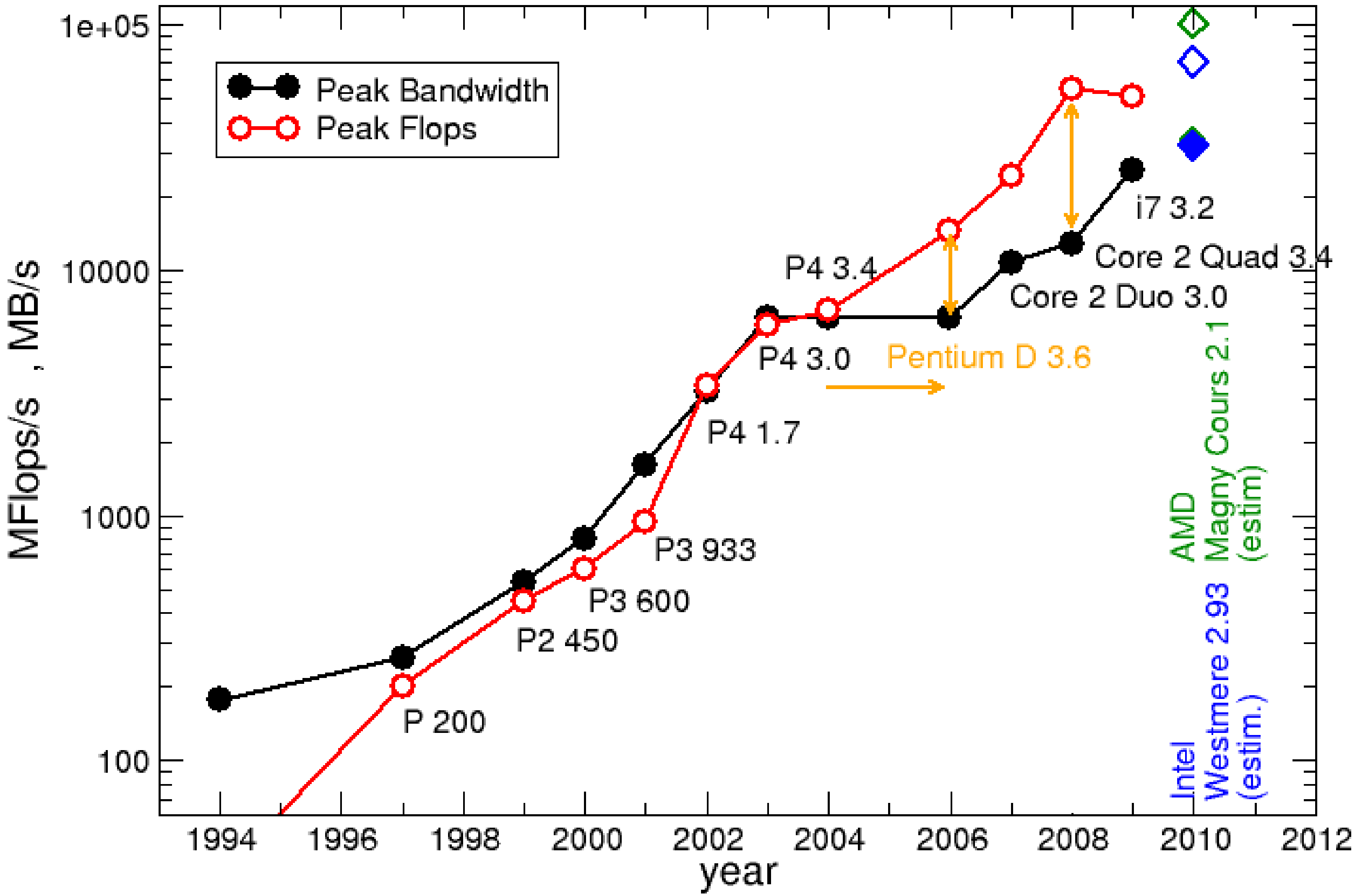
Test case	Runtime	Power	Energy
8 cores, plain C	90.43 s	89.69 Watt	8111 Joule
8 cores, SSE	29.63 s	92.62 Watt	2745 Joule
8 cores (SMT), SSE	22.61 s	102.07 Watt	2308 Joule
8 cores (SMT), AVX	18.42 s	110.80 Watt	2041 Joule

Test case	Runtime	Power	Energy
4 cores, plain C	154.72 s	55.61 Watt	8605 Joule
4 cores, SSE	49.99 s	58.01 Watt	2900 Joule
4 cores (SMT), SSE	-	-	-
4 cores (SMT), AVX	36.73 s	66.43 Watt	2440 Joule

Optimization pays out also with regard to energy!

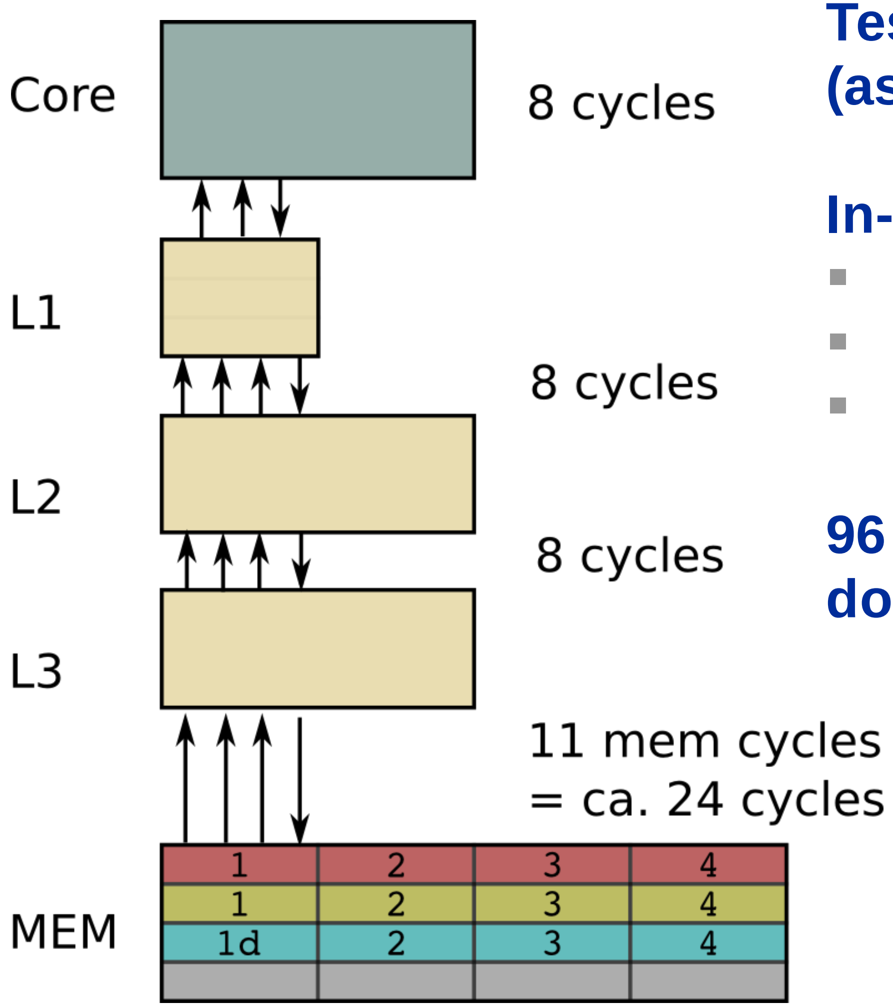
Where do we come from?

The Balance Metric



Case Study STREAM vector triad

Is it really that bad…



Test machine: Intel Nehalem gen (assume 3GHz)

In-memory runtime contributions:

- **17% Instruction execution**
- **33% In cache data transfers**
- **50% Memory data transfers**

96 GB/s / or 48 GB/s peak L1 BW doing nothing else than L/S

DEMO