

Erlangen Regional  
Computing Center



# Performance Engineering

Basic skills and knowledge



**Instrumentation based with gprof**

**Compile with `-pg` switch:**

```
icc -pg -O3 -c myfile1.c
```

**Execute the application. During execution a file `gmon.out` is generated.**

**Analyze the results with:**

```
gprof ./a.out | less
```

**The output contains three parts: A flat profile, the call graph, and an alphabetical index of routines.**

**The flat profile is what you are usually interested in.**

# Runtime profile with gprof: Flat profile

Time spent in routine itself

How often was it called

How much time was spent per call

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
66.86	26.14	26.14	502	0.05	0.05	ForceLJ::compute(Atom&, Neighbor&, Comm&, int)
30.77	38.17	12.03	26	0.46	0.46	Neighbor::build(Atom&)
1.43	38.73	0.56	1	0.56	38.46	Integrate::run(Atom&, Force*, Neighbor&, Comm&, Thermo&, Timer&)
0.36	38.87	0.14	2850	0.00	0.00	Atom::pack_comm(int, int*, double*, int*)
0.15	38.93	0.06	2850	0.00	0.00	Atom::unpack_comm(int, int, double*)
0.13	38.98	0.05	26	0.00	0.00	Atom::pbc()
0.10	39.02	0.04				__intel_sse3_rep_memcpy
0.08	39.05	0.03	25	0.00	0.00	Atom::sort(Neighbor&)
0.08	39.08	0.03	1	0.03	0.03	create_atoms(Atom&, int, int, int, double)
0.05	39.10	0.02	26	0.00	0.00	Comm::borders(Atom&)
0.00	39.10	0.00	1221559	0.00	0.00	Atom::pack_border(int, double*, int*)
0.00	39.10	0.00	1221559	0.00	0.00	Atom::unpack_border(int, double*)
0.00	39.10	0.00	131072	0.00	0.00	Atom::addatom(double, double, double, double, double, double)
0.00	39.10	0.00	1025	0.00	0.00	Timer::stamp(int)
0.00	39.10	0.00	502	0.00	0.00	Thermo::compute(int, Atom&, Neighbor&, Force*, Timer&, Comm&)
0.00	39.10	0.00	500	0.00	0.00	Timer::stamp()
0.00	39.10	0.00	475	0.00	0.00	Comm::communicate(Atom&)
0.00	39.10	0.00	26	0.00	0.00	Comm::exchange(Atom&)
0.00	39.10	0.00	25	0.00	0.00	Timer::stamp_extra_stop(int)
0.00	39.10	0.00	25	0.00	0.00	Timer::stamp_extra_start()
0.00	39.10	0.00	25	0.00	0.00	Neighbor::binatoms(Atom&, int)
0.00	39.10	0.00	7	0.00	0.00	Timer::barrier_stop(int)
0.00	39.10	0.00	1	0.00	0.00	create_box(Atom&, int, int, int, double)
0.00	39.10	0.00	1	0.00	0.00	create_velocity(double, Atom&, Thermo&)

Output is sorted according to total time spent in routine.

## Call executable with perf:

```
perf record -g ./a.out
```

## Analyze the results with:

```
perf report
```

## Advantages vs. gprof:

- Works on any binary without recompile
- Also captures OS and runtime symbols

```
Samples: 30K of event 'cycles:uppp', Event count (approx.): 20629160088
Overhead  Command          Shared Object          Symbol
 64.19%   miniMD-ICC       miniMD-ICC             [.] ForceLJ::compute
 31.54%   miniMD-ICC       miniMD-ICC             [.] Neighbor::build
  1.47%   miniMD-ICC       miniMD-ICC             [.] Integrate::run
  0.67%   miniMD-ICC       [kernel]               [k] irq_return
  0.40%   miniMD-ICC       miniMD-ICC             [.] Atom::pack_comm
  0.35%   mpiexec          [kernel]               [k] sysret_check
  0.21%   miniMD-ICC       miniMD-ICC             [.] create_atoms
  0.18%   miniMD-ICC       miniMD-ICC             [.] Atom::unpack_comm
  0.15%   miniMD-ICC       [kernel]               [k] sysret_check
  0.15%   miniMD-ICC       miniMD-ICC             [.] Comm::borders
  0.10%   miniMD-ICC       miniMD-ICC             [.] __intel_ssse3_rep_memcpy
  0.09%   miniMD-ICC       miniMD-ICC             [.] Atom::sort
  0.07%   miniMD-ICC       miniMD-ICC             [.] Neighbor::binatoms
```

Works out of the box for MPI/OpenMP parallel applications.

Example usage with MPI:

```
mpirun -np 2 amplxe-cl -collect hotspots -result-dir myresults -- a.out
```

- Compile with debugging symbols
- Can also resolve inlined C++ routines
- Many more collect modules available including hardware performance monitoring metrics

```
Elapsed Time: 8.650s
CPU Time: 8.190s
Effective Time: 8.190s
  Idle: 0.020s
  Poor: 8.170s
  Ok: 0s
  Ideal: 0s
  Over: 0s
  Spin Time: 0s
  Overhead Time: 0s
Total Thread Count: 2
Paused Time: 0s
```

```
Top Hotspots
Function                               Module      CPU Time
-----
ForceLJ::compute_fullneigh             miniMD-ICC  4.940s
Neighbor::build                         miniMD-ICC  2.820s
Integrate::finalIntegrate               miniMD-ICC  0.100s
Integrate::initialIntegrate             miniMD-ICC  0.060s
__intel_ssse3_rep_memcpy                miniMD-ICC  0.040s
[Others]                                N/A         0.230s
```

Performance measurements have to be accurate, deterministic and reproducible.

Components for application benchmarking:

Timing

Documentation

Affinity control

System  
configuration

Always run benchmarks on an **EXCLUSIVE SYSTEM!**

For benchmarking, an accurate wallclock timer (end-to-end stop watch) is required:

- `clock_gettime()`, POSIX compliant timing function
- `MPI_Wtime()` and `omp_get_wtime()`, standardized programming-model-specific timing routines for MPI and OpenMP

```
#include <stdlib.h>
#include <time.h>
```

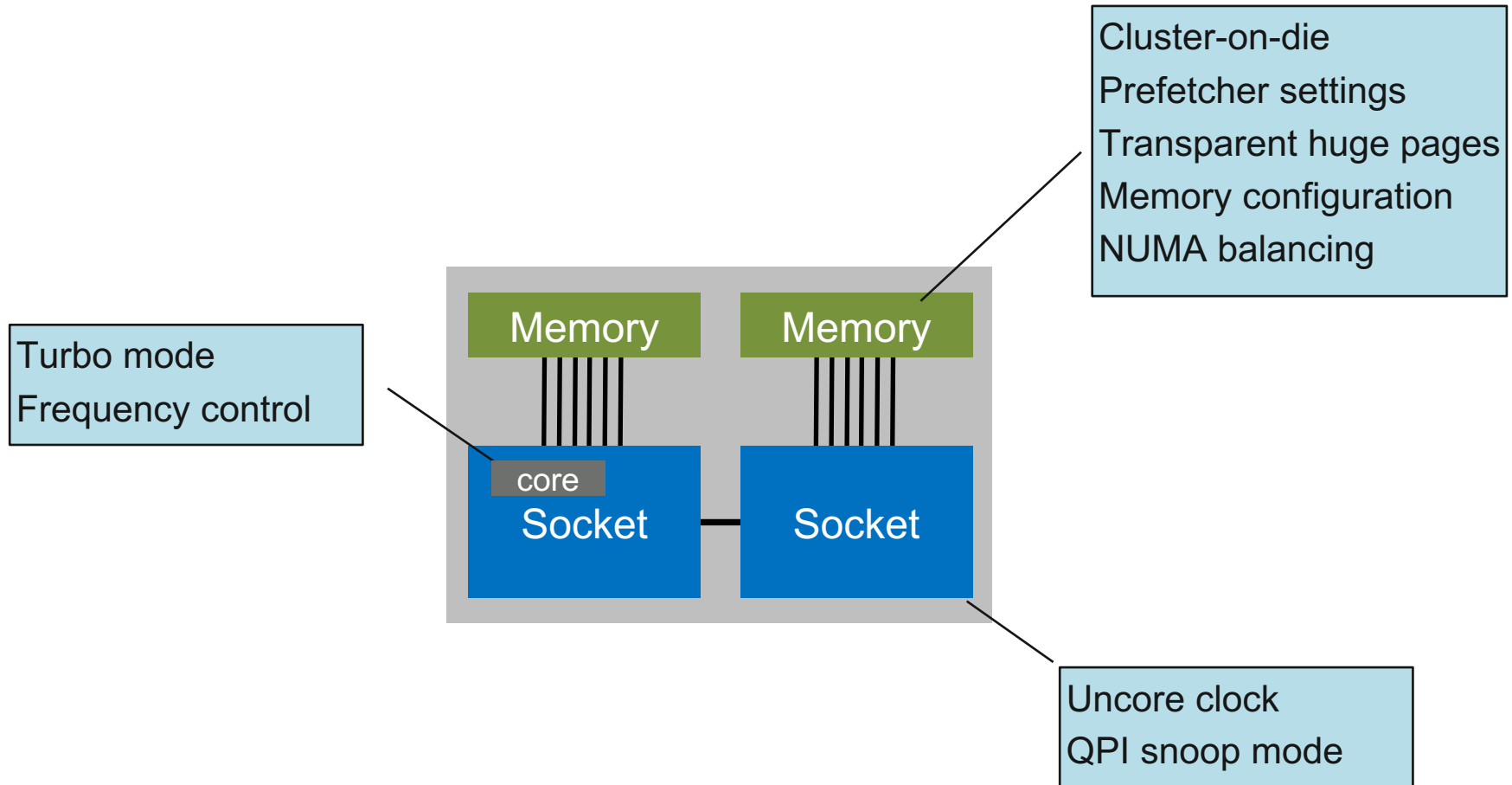
```
double getTimeStamp()
{
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return (double)ts.tv_sec + (double)ts.tv_nsec * 1.e-9;
}
```

## Usage:

```
double S, E;
S = getTimeStamp();
/* measured code region */
E = getTimeStamp();
return E-S;
```



<https://github.com/RRZE-HPC/TheBandwidthBenchmark/>



Shell script for system state dump (requires Likwid tools):

<https://github.com/RRZE-HPC/ThePerformanceLogbook/blob/master/Template/helpers/machine-state.sh>



## Query turbo mode steps with `likwid-powermeter -i`

```
-----  
CPU name:      Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz  
CPU type:      Intel Skylake SP processor  
CPU clock:     2.40 GHz  
-----
```

```
Base clock:    2400.00 MHz  
Minimal clock: 1000.00 MHz
```

### Turbo Boost Steps:

```
C0 3700.00 MHz  
C1 3700.00 MHz  
C2 3500.00 MHz  
C3 3500.00 MHz  
C4 3400.00 MHz  
C5 3400.00 MHz  
C6 3400.00 MHz  
C7 3400.00 MHz
```

```
C17 3100.00 MHz  
C18 3100.00 MHz  
C19 3100.00 MHz
```

### Info about Uncore:

```
Minimal Uncore frequency: 1200 MHz  
Maximal Uncore frequency: 2400 MHz
```

Above steps are valid for scalar or SSE code; refer to vendor docs for AVX and AVX515 steps.

## Query and set frequencies with `likwid-setFrequencies`

### Current CPU frequencies:

```
CPU 0: governor      ondemand min/cur/max 1.0/1.000/2.401 GHz Turbo 0  
CPU 1: governor      ondemand min/cur/max 1.0/1.001/2.401 GHz Turbo 0  
CPU 2: governor      ondemand min/cur/max 1.0/1.004/2.401 GHz Turbo 0  
CPU 5: governor      ondemand min/cur/max 1.0/1.000/2.401 GHz Turbo 0  
CPU 6: governor      ondemand min/cur/max 1.0/1.000/2.401 GHz Turbo 0
```

```
CPU 63: governor     ondemand min/cur/max 1.0/1.004/2.401 GHz Turbo 0  
CPU 64: governor     ondemand min/cur/max 1.0/1.001/2.401 GHz Turbo 0  
CPU 67: governor     ondemand min/cur/max 1.0/1.000/2.401 GHz Turbo 0  
CPU 68: governor     ondemand min/cur/max 1.0/1.003/2.401 GHz Turbo 0  
CPU 69: governor     ondemand min/cur/max 1.0/1.000/2.401 GHz Turbo 0  
CPU 73: governor     ondemand min/cur/max 1.0/1.344/2.401 GHz Turbo 0  
CPU 74: governor     ondemand min/cur/max 1.0/1.285/2.401 GHz Turbo 0  
CPU 77: governor     ondemand min/cur/max 1.0/1.028/2.401 GHz Turbo 0  
CPU 78: governor     ondemand min/cur/max 1.0/1.059/2.401 GHz Turbo 0  
CPU 79: governor     ondemand min/cur/max 1.0/1.027/2.401 GHz Turbo 0
```

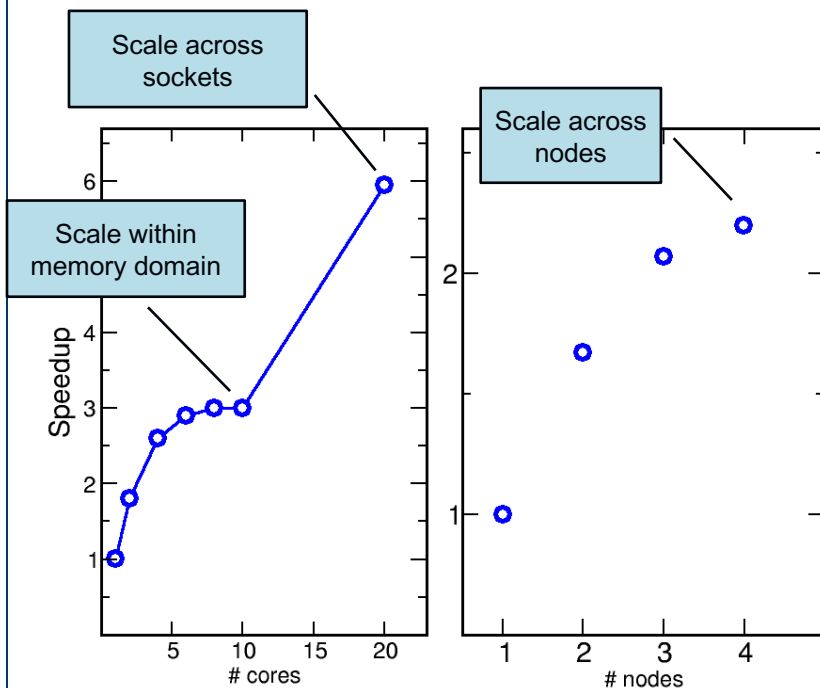
### Current Uncore frequencies:

```
Socket 0: min/max 1.2/2.4 GHz  
Socket 1: min/max 1.2/2.4 GHz
```

Always measure the actual frequency with a HPM tool. Using `likwid-perfctr`, any performance group reports clock frequencies.

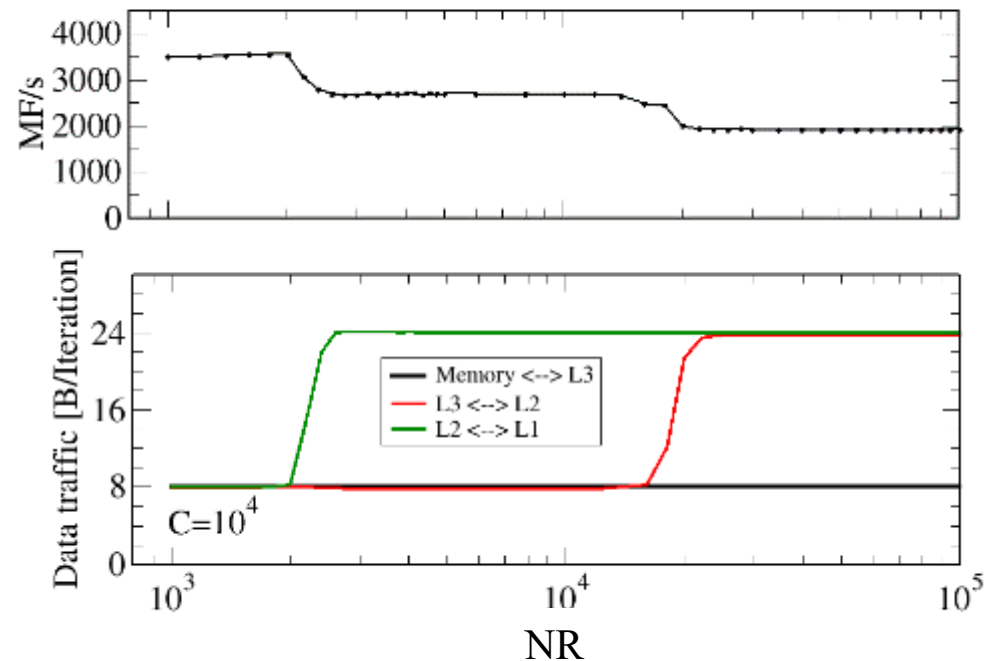
Two main variations:

## Core count



Choosing the right scaling baseline

## Dataset size



- Measure with one process (to start with)
- Scan dataset size in fine steps
- Verify the data volumes with a HPM tool

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**

- Manual and knowledge collection how to build, configure and run application
- Document activities and results in a structured way
- Learn about best practice guidelines for performance engineering
- Serve as a well defined and simple way to exchange and hand over performance projects

The **logbook** consists of a single **markdown document**, helper scripts, and directories for input, raw results, and media files.



<https://github.com/RRZE-HPC/ThePerformanceLogbook>

- Generic HPC-specific documentation Wiki
- Open for participation, login using SSO via eduGAIN
- Covers specifically performance engineering topics and skills
- Still work in progress, but improving!

<https://hpc-wiki.info>