NHR FAU

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

MuCoSim Introduction (Part II)

Thomas Gruber and Katrin Nusser (HPC @ Uni Erlangen)

Thomas.Gruber@fau.de

HPC High Performance Computing

Recap from last week

Get a fresh interactive job on the Meggie cluster with hardware performance monitoring enabled

Open a second terminal to Meggie frontend for compilation

What are the topological entities of a common HPC node nowadays?

What is the command to submit a non-interactive job to the cluster?

Which command can be used to get a recent compiler?

How are the general optimization levels of compilers named?

If the performance of my application varies a lot, what could be the reason?

How many affinity domains does your system provide? (`--help`)

Compile `pin/hello_pthread.c` (`-pthread`)
Run it a few times, how often do threads share a CPU?

Pin `hello_pthread` (5 threads)

Pin `hello_pthread` to the first two physical HW threads of all sockets

What happens? Who wins?

```
OMP_NUM_THREADS=10 likwid-pin -c 0-4 ./hello_pthread
```

Run `stream` with 4 threads pinned differently (`N:0-3`, `S0:0-1@S1:0-1`).
What‘s the fastest CPU selection for `triad`?

NHR FAU

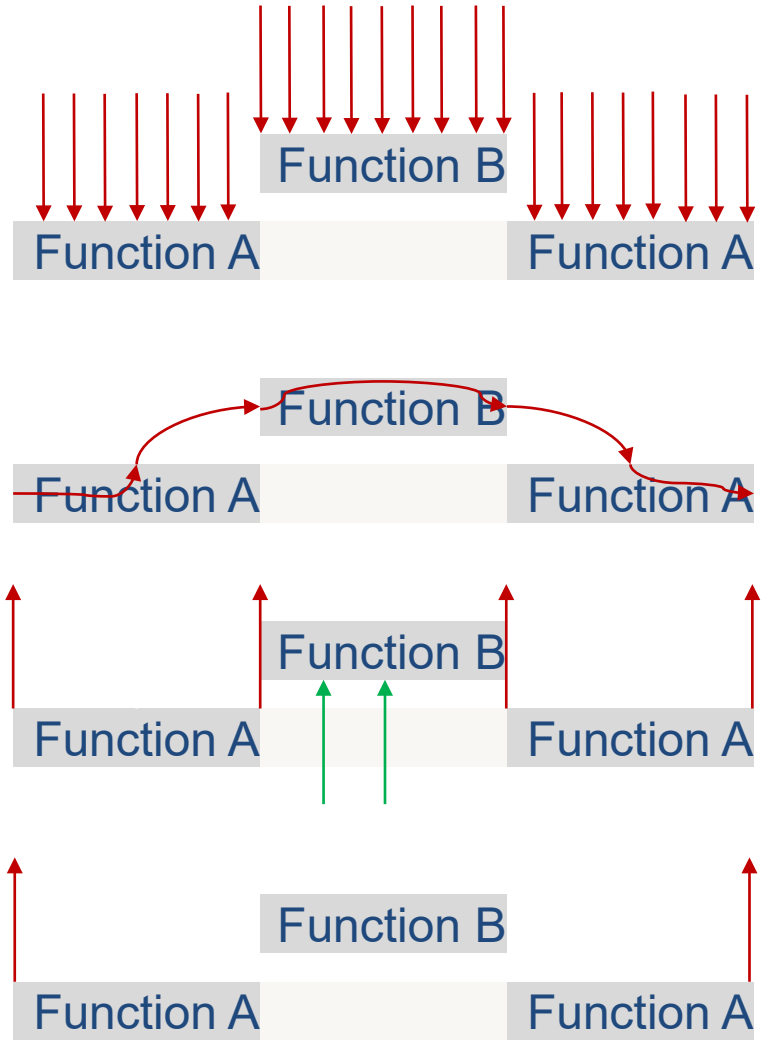
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

MuCoSim Introduction

Analysis of applications

Measurement techniques

- Sampling
- Tracing
- Instrumentation
- Profiling



gprof, xray, perf, …

- Read state periodically
- Follow the execution path
- Add `read_state()` where desired
- Create an overview what happened

Runtime profile - Find out hotspots in the code

- Many tools available: **`gprof`**, **`xray`**, **`perf`**, …
- For **`gprof`** build with **`-pg`**
- Creates XML and tabular output files with fields:
  - Time and time share for function
  - Call and exit count
  - File and line of function
- Run application like normal
- Afterwards: **`gprof <exec> gmon.out`**

Runtime profile

Inclusive timing

Exclusive timing

Time(%)	Self (sec)	Call count	Function
44.52	2.47	100	`triad()`
25.96	1.44	100	`add()`
16.94	0.94	100	`copy()`
100.00	0.00	1	`main()`

How often a function was called

Besides function runtime, how can we measure resource usage?

So, how to restrict measurements to the `triad()` function?

Compile **`stream`** in **`runtime_profile`** with runtime profiling
(use *.c and include header path **`-I.`**)

Look at the hotspots in the code.
Can you name a reason for the runtime difference?

What is hardware performance monitoring?
Overview about HPM

- Performance monitoring units (PMUs) at hardware level
- Introduced for x86 with Intel Pentium (1994)
- Originally used by CPU vendors for hardware validation
- No additional CPU work to handle hardware events in PMUs
- Accessing PMUs requires CPU work → Overhead
- Limited number of counters per PMU (x86: 4 per unit)

Hardware Performance Monitoring with LIKWID - `likwid-perfctr`

- **`likwid-perfctr`** sets up system topology and perfmon
- Setup, start, read and stop PMUs
- Execute application on given CPU set (**`-C`**)
- Evaluate counter values

```
likwid-perfctr -C 0 -g INST_RETIRED_ANY:FIXC0 <app>
```

- LIKWID needs you to specify which counter runs which event
- Combine multiple (event+counter)s with ‘,’
- For advanced usage, the events can be enriched with options **`threshold`**, **`invert`**, **`count_kernel`**, **`edge_detect`**,

LIKWID - HPM with `likwid-perfctr`

```
$ likwid-perfctr -C 0,1 -g L2_TRANS_L1D_WB:PMC0 ./app
+-----------------------+---------+-------------+-------------+
|         Event         | Counter |    Core 0   |    Core 1   |
+-----------------------+---------+-------------+-------------+
| Runtime (RDTSC) [s]   |   TSC   | 2.573182e+00 | 2.573182e+00 |
|   L2_TRANS_L1D_WB     |  PMC0   |   281176518 |   281240170 |
+-----------------------+---------+-------------+-------------+
```

- Event names (in many cases) not intuitive
- Events are architecture-specific
- Some sound promising but return bad counts, others are broken
- More interest in real metrics like volume of loaded/stored data

LIKWID - HPM with `likwid-perfctr`

- LIKWID defines performance groups
  ≈ eventlist + derived metrics + documentation
- List all groups: **`likwid-perfctr -a`**

You can also define own performance groups!

```
$ likwid-perfctr -C 0,1 -g L2 ./app
+-------------------------------+-----------+-----------+
|             Metric            |   Core 0  |   Core 1  |
+-------------------------------+-----------+-----------+
|       Runtime (RDTSC) [s]     |    2.6439 |    2.6439 |
|  L2D load bandwidth [MBytes/s]  | 6744.8121 | 6743.6037 |
|  L2D load data volume [GBytes]  |   17.8325 |   17.8293 |
|  L2D evict bandwidth [MBytes/s] | 3372.4061 | 3371.8019 |
|  L2D evict data volume [GBytes] |    8.9163 |    8.9147 |
```

LIKWID - Performance groups

- **FLOPS_AVX**: Packed AVX MFlops/s
- **FLOPS_DP**: Double Precision MFlops/s
- **FLOPS_SP**: Single Precision MFlops/s
- **DATA**: Load to store ratio
- **L2**: L2 cache bandwidth in MBytes/s
- **L3**: L3 cache bandwidth in MBytes/s
- **MEM**: Main memory bandwidth in MBytes/s
- **ENERGY**: Power and Energy consumption
- **MEM_DP**: Memory & DP FLOP/s & Energy
- **MEM_SP**: Memory & SP FLOP/s & Energy

Hardware Performance Monitoring with LIKWID - `likwid-perfctr`

- **`$ likwid-perfctr -C 0,1 -g FLOPS_DP ./a.out`**

  Measure DP FLOP/s of the whole application run of on CPUs 0, 1
- **`$ likwid-perfctr -c 0,1 -g DATA ./a.out`**

  Measure load/store ratio on CPUs 0,1. Application is not pinned!
- **`$ likwid-perfctr -g MEM_DP -H`**

  Get help for performance group MEM_DP
- **`$ likwid-perfctr -e (| less)`**

  List all events and counters, search with **`-E <searchstr>`**

Compile **`perfctr/triad.c`**
Measure the memory bandwidth (**`MEM`**)
from 1 to number of phys. cores per socket
At which core count does it saturate?

Compile **`perfctr/pi.c`**
Measure the FLOP rate from 4 to 10 processes on one socket
Does it have a saturation point?
How well is it vectorized? What is the max. vectorization ratio you can achieve? Are all operations done with „best“ vectorization?

LIKWID - *HPM of functions*

- LIKWID offers MarkerAPI for code region measurements

```
#include <likwid-marker.h>
LIKWID_MARKER_INIT; // in serial region
LIKWID_MARKER_REGISTER("Compute"); // in parallel region
LIKWID_MARKER_START("Compute");
<code>
LIKWID_MARKER_STOP("Compute");
LIKWID_MARKER_CLOSE;  // in serial region
```

Older version use **<likwid.h>**

Reduces startup overhead

Multiple regions and nesting allowed

- Compile with **-DLIKWID_PERFMON**

Add marker API to code (restructure loops)

```
#pragma omp parallel for
    <loop>
```

```
#pragma omp parallel
{
LIKWID_MARKER_START("Compute");
#pragma omp for
    <loop>

LIKWID_MARKER_STOP("Compute");
}
```

Add marker API to code (closed-source library calls)

```
some_parallel_f()
```

```
#pragma omp parallel
{
    LIKWID_MARKER_START("foo")
}
some_parallel_f()
#pragma omp parallel
{
    LIKWID_MARKER_STOP("foo")
}
```

LIKWID - *HPM of functions*

Compile @ RRZE:

Defined by LIKWID module at RRZE

```
$CC -DLIKWID_PERFMON $LIKWID_INC $LIKWID_LIB code.c -o code -llikwid
```

```
likwid-perfctr -C <cpustr> -g <group> -m ./a.out
```

Use capital C
MarkerAPI requires pinned threads

Tells `likwid-perfctr` to use MarkerAPI mode

Copy **`perfctr/pi.c`** to **`marker/pi.c`**

Add MarkerAPI calls around loop for each OpenMP thread & compile

Measure the FLOP rate from 4 to number of phys. cores per socket

Compile **`marker/stream.c`** (use **`-I. *.c`**)

What are the read and write memory bandwidths of each hotspot for 4 threads?

Compare results to the application output of stream.

Is there a difference and if yes, why?

NHR FAU

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Thank you for your attention!

Erlangen National High Performance Computing Center (NHR@FAU)

Martensstraße 1, 91058 Erlangen

http://www.rrze.fau.de

Thomas.Gruber@fau.de