A Topology-Aware Performance Monitoring Tool for Shared Resource Management in Multicore Systems

TADaaM Team - Nicolas Denoyelle - Brice Goglin - Emmanuel Jeannot

August 24, 2015

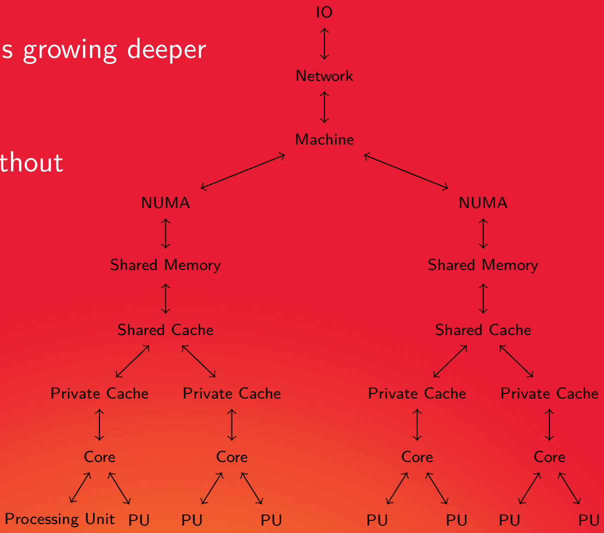
1. Context/Motivations
2. Fast presentation of the tool
3. Demonstration
4. How does it works ?
5. How is it made ?
6. Features & Future Works

MOTIVATIONS

Memory hierarchy is growing deeper and larger.

No performance without a fair usage of the system topology

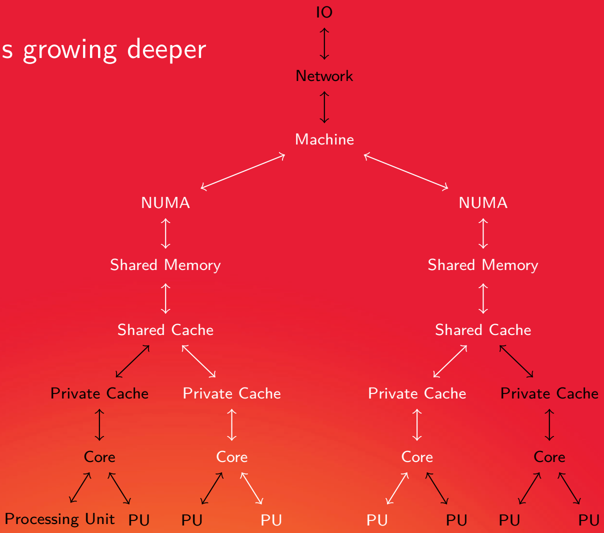
Batch schedulers, runtimes, applications themeselves . . . are getting topology aware.



MOTIVATIONS

Memory hierarchy is growing deeper and larger.

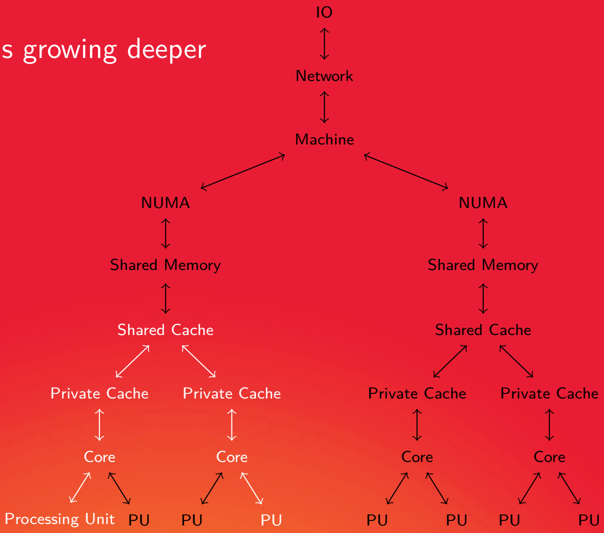
Hence, data management gives opportunities for performance improvements.



MOTIVATIONS

Memory hierarchy is growing deeper and larger.

Hence, data management gives opportunities for performance improvements.

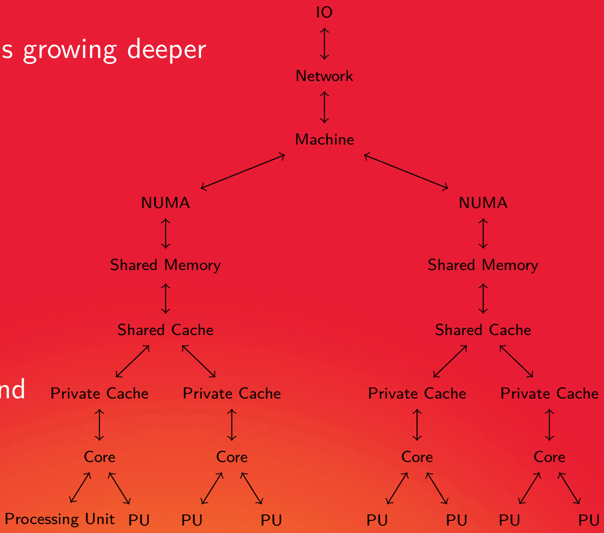


MOTIVATIONS

Memory hierarchy is growing deeper and larger.

Hence, data management gives opportunities for performance improvements.

It is a multi-level and a multi-criteria problem.



MOTIVATIONS

- Need to match use cases, and relevant performance metrics for each level.
- **Need to match performance and topology.**

- Requires topology modeling skills.
- Requires adaptable performance monitoring.

Yet Another Tool to Monitor Applications Performance

- Focus on data presentation to link the results with topology informations.

- Relies on two cornerstones of topology modeling (hwloc) and performance counter abstraction (PAPI) to map the latter on the former

- Minimal configuration and software requirements.

- Can help finding and caracterizing localized bottlenecks.

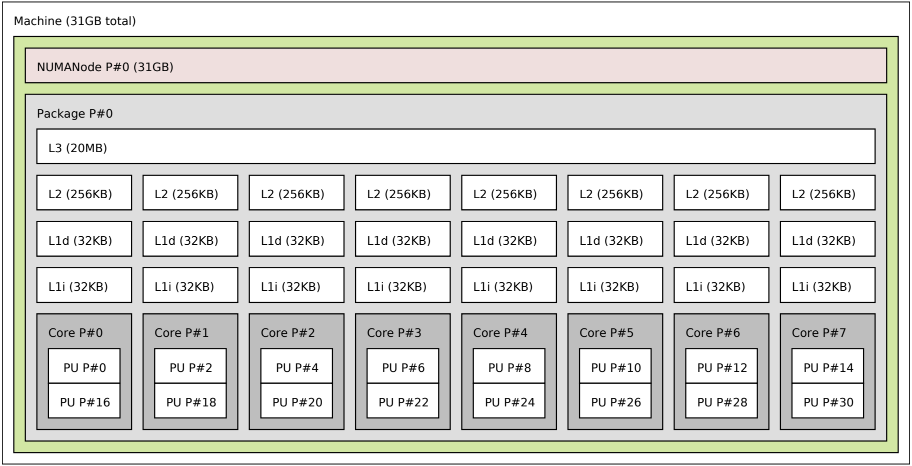
Hardware Locality (hwloc)

Portable abstraction of hierarchical architectures for high-performance computing

- Performs topology discovery and extracts hardware component information.
- Provides tools for memory and process binding.
- Many operating systems supported
- ...
- lstopo utility to display the topology:

Developped at Inria Bordeaux.

Hardware Locality (hwloc)



Machine (31GB total)

NUMANode P#0 (31GB)

Package P#0

L3 (20MB)

L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)

L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)

L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)

Core P#0: PU P#0, PU P#16

Core P#1: PU P#2, PU P#18

Core P#2: PU P#4, PU P#20

Core P#3: PU P#6, PU P#22

Core P#4: PU P#8, PU P#24

Core P#5: PU P#10, PU P#26

Core P#6: PU P#12, PU P#28

Core P#7: PU P#14, PU P#30

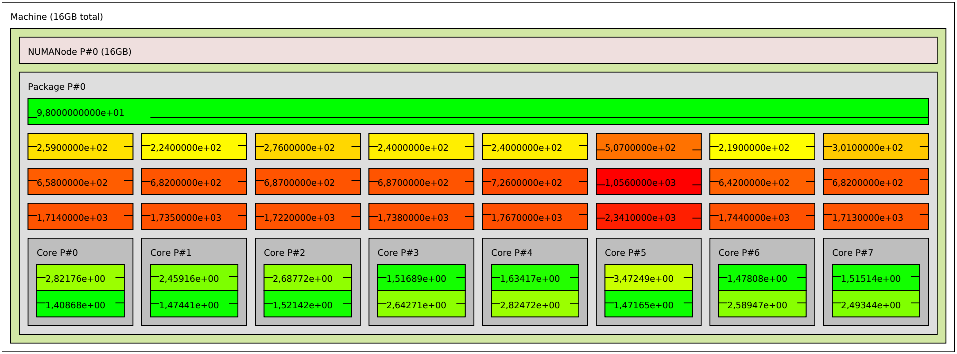
Performance Application Programming Interface (PAPI)

Consistent interface and methodology for use of the performance counter hardware.

- Real time relation between software performance and processor events.
- Many operating systems supported too.
- Reliable and actively supported.
- Used in a wide range of performance analysis applications.

An abstraction layer to plug some other performance library is under development.

Dynamic Lstopo (example)



Machine (16GB total)

NUMANode P#0 (16GB)

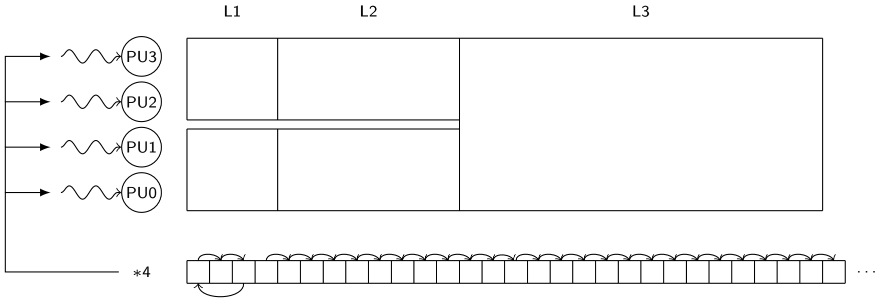
Package P#0

9,8000000000e+01

2,5900000e+02	2,2400000e+02	2,7600000e+02	2,4000000e+02	2,4000000e+02	5,0700000e+02	2,1900000e+02	3,0100000e+02
6,5800000e+02	6,8200000e+02	6,8700000e+02	6,8700000e+02	7,2600000e+02	1,0560000e+03	6,4200000e+02	6,8200000e+02
1,7140000e+03	1,7350000e+03	1,7220000e+03	1,7380000e+03	1,7670000e+03	2,3410000e+03	1,7440000e+03	1,7130000e+03
Core P#0	Core P#1	Core P#2	Core P#3	Core P#4	Core P#5	Core P#6	Core P#7
2,82176e+00	2,45916e+00	2,68772e+00	1,51689e+00	1,63417e+00	3,47249e+00	1,47808e+00	1,51514e+00
1,40868e+00	1,47441e+00	1,52142e+00	2,64271e+00	2,82472e+00	1,47165e+00	2,58947e+00	2,49344e+00

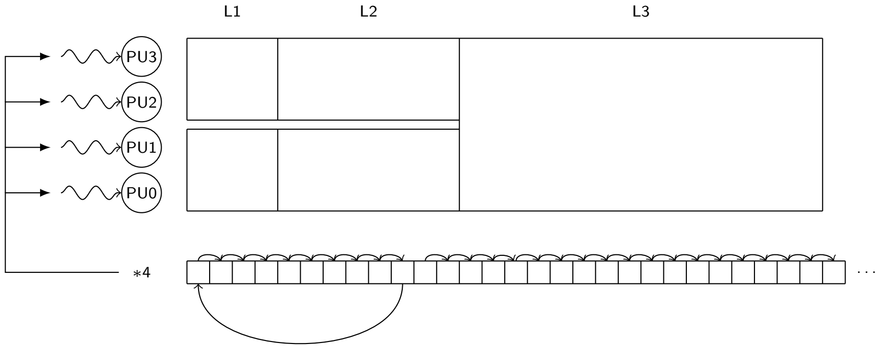
Sample of hardware performance counters mapped on a single socket of an Intel Xeon E5-2650 CPU.

A Demonstration Worth Thousand Words



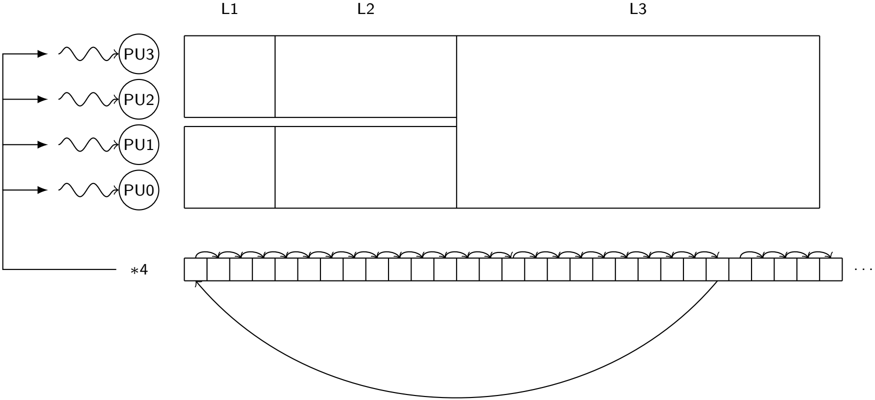
Accesses to a linked list of variable size.

A Demonstration Worth Thousand Words



Accesses to a linked list of variable size.

A Demonstration Worth Thousand Words



Accesses to a linked list of variable size.

A Demonstration Worth Thousand Words

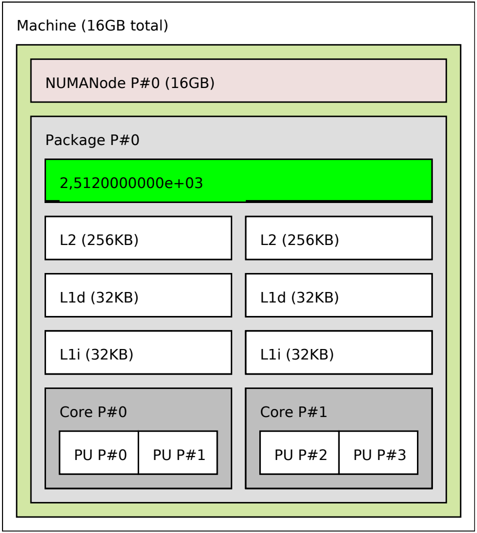
```
L3_MISS{
  OBJ = L3;
  CTR = PAPI_L3_TCM;
  LOGSCALE = 1;
}

L2_MISS{
  OBJ = L2;
  CTR = PAPI_L2_TCM;
  LOGSCALE = 1;
}
```

```
L1_MISS{
  OBJ = L1d;
  CTR = PAPI_L1_DCM;
  LOGSCALE = 1;
}

SINGLE_L3_MISS{
  OBJ = PU;
  CTR = PAPI_L3_TCM;
  LOGSCALE = 1;
}
```

Dynamic Lstopo (Usage)



Counters input:

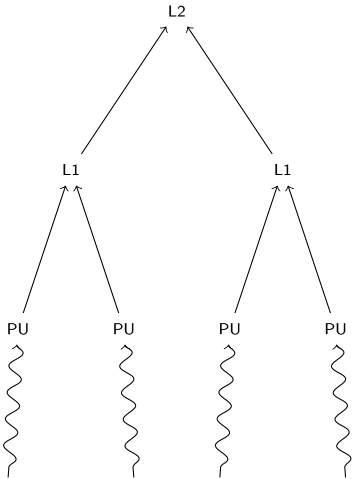
```
SINGLE_L3_MISS{
   OBJ = L3;
   CTR =
   PAPI_L2_TCM/PAPI_L2_TCA;
   LOGSCALE = 1;
   MAX=1000000;
   MIN=0;
}
```

Command line:

```
lstopo –perf-input counters_input
```

Dynamic Lstopo (Theory)

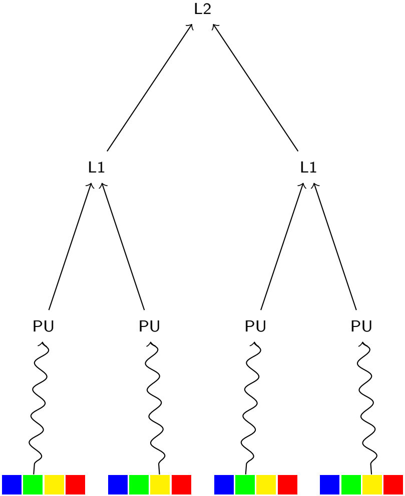
1. Spawn one pthread per hardware thread (PU#0, . . . , PU#3).



Dynamic Lstopo (Theory)

1. Spawn one pthread per hardware thread (PU#0, . . . , PU#3).

2. For each timestamp, each thread collects a local set of performance counters.

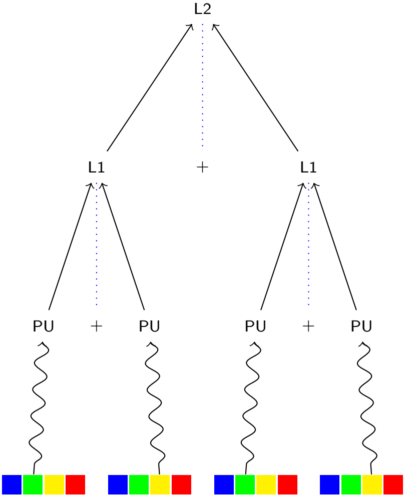


Dynamic Lstopo (Theory)

1. Spawn one pthread per hardware thread (PU#0, . . . , PU#3).

2. For each timestamp, each thread collects a local set of performance counters.

3. Counters are accumulated in each upper level.



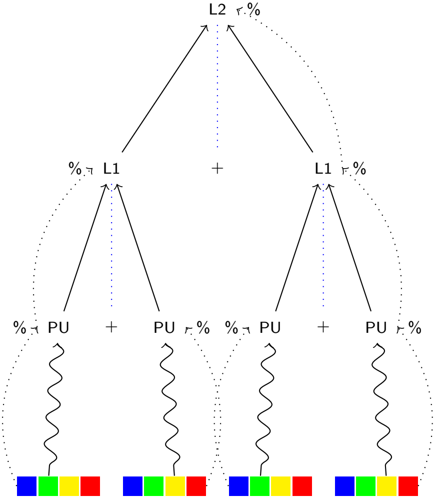
Dynamic Lstopo (Theory)

1. Spawn one pthread per hardware thread (PU#0, . . . , PU#3).

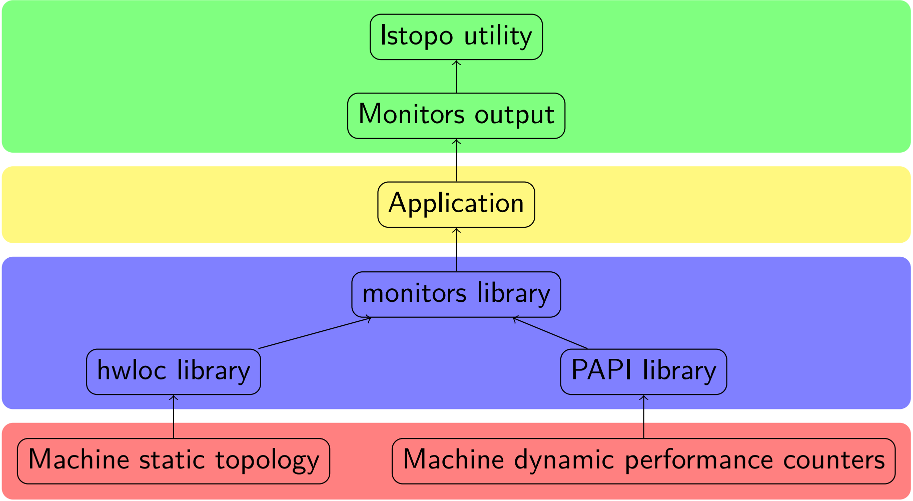
2. For each timestamp, each thread collects a local set of performance counters.

3. Counters are accumulated in each upper level.

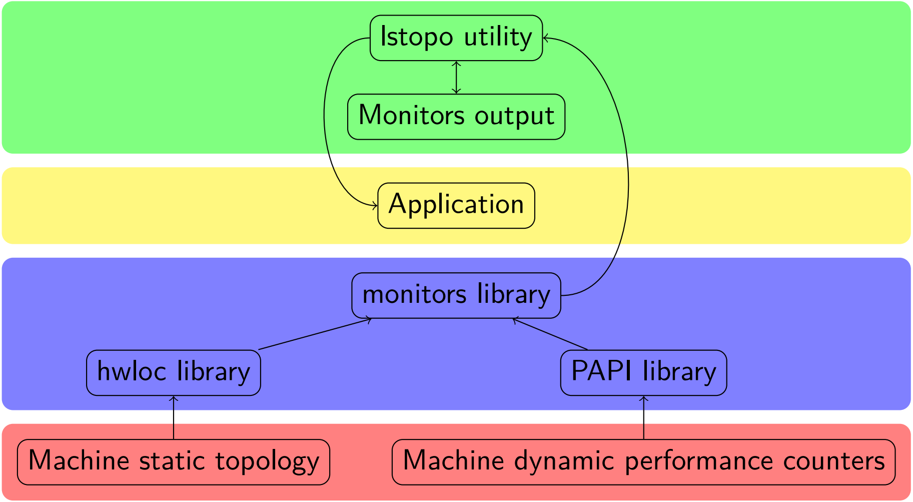
4. For each level, a leaf computes an arithmetic expression of the performance counters in the set.



Dynamic Lstopo Software Architecture in Brief



Dynamic Lstopo Software Architecture in Brief



API

```
monitors = load_Monitors_from_config(NULL,
    "my_perf_file", "my_output_file", 0);

Monitors_watch_pid(monitors,getpid());

Monitors_start(monitors);

/* ... */

Monitors_update_counters(monitors);

delete_Monitors(monitors);
```

Dynamic Lstopo (Output paje trace)

```
%EventDef val 0
%    Id         int
%    Phase      int
%    Time_us    date
%    Value      double
%EndEventDef

%EventDef container 1
%    Id         int
%    Level      string
%    Sibling    int
%    Name       string
%    Logscale   int
%EndEventDef
```

```
1 3 PU 0 SINGLE_L3_MISS 1
1 2 PU 1 SINGLE_L3_MISS 1
1 1 PU 2 SINGLE_L3_MISS 1
1 0 PU 3 SINGLE_L3_MISS 1

0 2 0 962832762224 67,00
0 1 0 962832762224 58,00
0 3 0 962832762225 77,00
0 0 0 962832762236 64,00
0 3 0 962832860676 94514,00
0 0 0 962832860676 121746,00
0 2 0 962832860676 205170,00
0 1 0 962832860717 200931,00
```

Features

- Record and/or Display live machine performance counters and match them with topology.

- Several settings: counters accumulation, sampling rate, attach to a process. . .

- Replay any trace with a topology file (for external display)..

- Sample specific parts of an application with the monitor library.

- Support legacy lstopo options (restrict topology, change display format. . .).

Future works

- Match code and performance informations
- Accept user defined aggregation operator.
- Provide performance abstraction layer.
- Be able to delimit phases during execution.
- Find and give explicit hints on bottlenecks.

Conclusion

Data locality becomes a main criterion for high performance.

We built a tool based on a topology model and a performance library to help taking up the challenge.

- It maps performance values to machine objects.
- It is a visual tool, fast and easy to use.
- It is lightweight and causes less than 1% CPU overhead.
- Let you build topology aware performance models.

Dynamic lstopo is into the process of beeing merged with hwloc project.

Now available from https://github.com/NicolasDenoyelle/dynamic_lstopo

Thank you