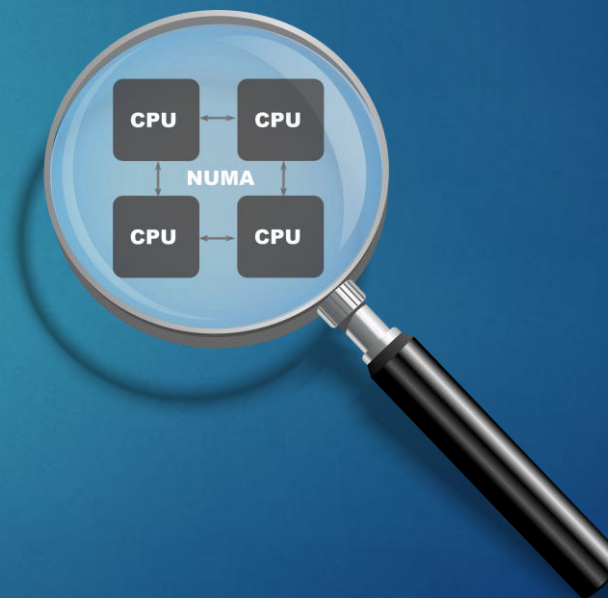


NUMAPROF

A NUMA MEMORY PROFILING TOOL



PLAN

2

Sébastien Valat
15/02/2018

- ▶ Reminder & what we want
- ▶ NUMAPROF internals
- ▶ GUI and example
- ▶ Conclusion

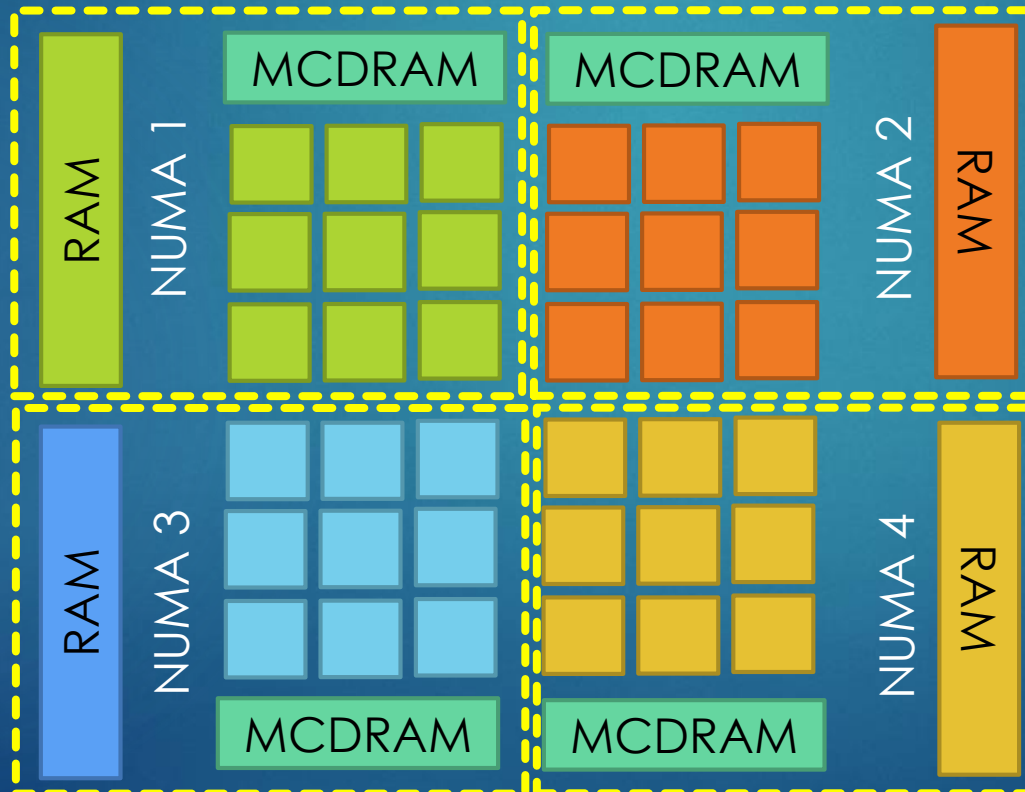
Reminder on NUMA

Today topology

4

Sébastien Valat
15/02/2018

- ▶ Example current **Intel Knight Landing**, mode **SNC2** or **SNC4**
- ▶ Also add fast memory **MCDRAM** presented as **NUMA** or **LLC cache**

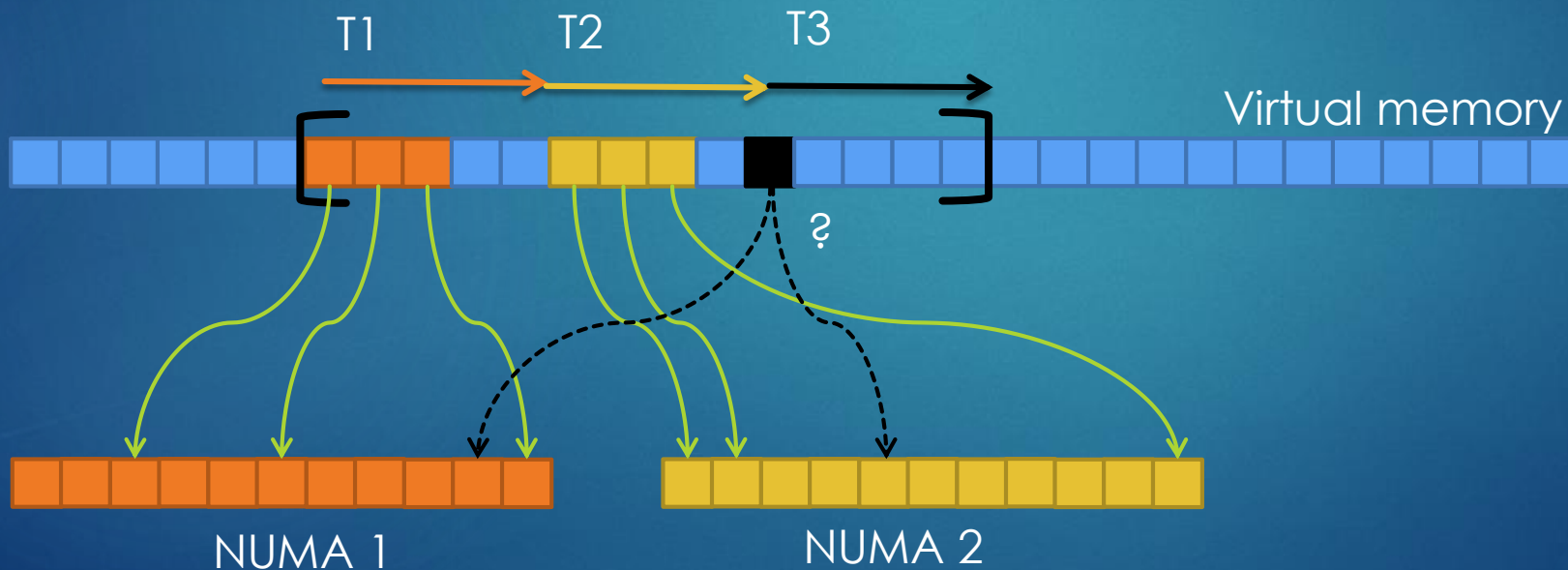


Implicit binding : first touch

5

Sébastien Valat
15/02/2018

- ▶ New allocated segments are **physically empty**
- ▶ They are filled on **first touch**
- ▶ Page selection **depend** of the **thread position**



Typical OpenMP mistake

6

Sébastien Valat
15/02/2018

- ▶ Make first **init outside of OpenMP** (in thread 1)
- ▶ So **each pages** will be first touched **on NUMA 1**

```
#pragma omp parallel for
```

```
for (int i = 0 ; i < SIZE ; i++)  
    array[i] = 0;
```

- ▶ Then access

```
#pragma omp parallel for
```

```
for (int i = 0 ; i < SIZE ; i++)  
    array[i]++;
```

- ▶ **Bad performance** due to remote accesses !

Wish list for a profiling tool...

7

Sébastien Valat
15/02/2018



- ▶ We want to know if we make **remote accesses**
- ▶ Ideally we need to know **where...**
- ▶ We can dream, we want to know **which allocation contain issues**
- ▶ We want to know **where** the **first touch** has been done
- ▶ On KNL we want to check **MCRAM accesses**

8

Sébastien Valat
15/02/2018

- The screenshot displays the LULESH 3D application interface, which is divided into three main sections:

 - Source Code (Top Left):** Shows the `LULESH_3D.noGlobal.TALC.C.bak.C` file. The code defines a `new[]` operator for `planeEdgeNodes` and `rowEdgeNodes` arrays, which are used to store node indices for a given plane and row. The code is as follows:


```

2483 for (Index_t plane=0; plane<edgeNodes; ++plane) {
2484     ty = Real_t(0.);
2485     for (Index_t row=0; row<edgeNodes; ++row) {
2486         tx = Real_t(0.);
2487         for (Index_t col=0; col<edgeNodes; ++col) {
2488             x[idxs] = tx;
2489             y[idxs] = ty;
2490             z[idxs] = tz;
      
```
 - Plot (Top Right):** A line plot titled "[Plot graph] operator new[] (unsigned long): LOW OFFSET (t)". The x-axis is labeled "t (hr)" and ranges from 0 to 45. The y-axis is labeled "y Value" and ranges from 0.0E00 to 1.0E00. The plot shows a single data series that starts at approximately 0.5 and remains relatively constant, with a slight upward trend towards the end of the time range.
 - Calling Context View (Bottom):** A table showing the execution context and various metrics. The table is organized into two main sections: "special metrics" and "common metrics".

| Scope | NUMA_LATENCY:Sum (t) | ipi_NUMA | NUMA_MATCH:Sum (t) | NUMA_MISMATCH:Sum (t) | NUMA_NODE0:Sum (t) |
|--|----------------------|----------|--------------------|-----------------------|--------------------|
| Experiment Aggregate Metrics | 4.81e+06 100 % | 4.66e-01 | 3.14e+06 100 % | 1.41e+06 100 % | 1.53e+06 100 % |
| ↳ monitored_heap_data | 3.57e+06 74.24 % | 1.17e-01 | 1.05e+06 3.34 % | 1.99e+05 14.14 % | 2.37e+05 15.54 % |
| ↳ ↳ #267: heap_data_allocation | 3.57e+06 74.24 % | 1.17e-01 | 1.05e+05 3.34 % | 1.99e+05 14.14 % | 2.37e+05 15.54 % |
| ↳ ↳ ↳ #297: monitor_main | 3.57e+06 74.24 % | 1.17e-01 | 1.05e+05 3.34 % | 1.99e+05 14.14 % | 2.37e+05 15.54 % |
| ↳ ↳ ↳ ↳ #479: main | 3.57e+06 74.24 % | 1.17e-01 | 1.05e+05 3.34 % | 1.99e+05 14.14 % | 2.37e+05 15.54 % |
| ↳ ↳ ↳ ↳ ↳ #2160: operator new(unsigned long) | 5.45e+05 11.34 % | 1.60e-01 | 4.66e+03 0.14 % | 2.95e+04 2.14 % | 3.40e+04 2.24 % |
| ↳ ↳ ↳ ↳ ↳ ↳ #32: operator new(unsigned long) | 5.45e+05 11.34 % | 1.60e-01 | 4.66e+03 0.14 % | 2.95e+04 2.14 % | 3.40e+04 2.24 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ #52: malloc | 5.45e+05 11.34 % | 1.60e-01 | 4.66e+03 0.14 % | 2.95e+04 2.14 % | 3.40e+04 2.24 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #292: heap_data_accesses | 5.45e+05 11.34 % | 1.60e-01 | 4.66e+03 0.14 % | 2.95e+04 2.14 % | 3.40e+04 2.24 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #232: gomp_thread_start | 5.45e+05 11.34 % | 1.60e-01 | 3.74e+03 0.14 % | 2.32e+04 2.14 % | 3.32e+04 2.24 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #204: _ZL28CalcFibonacciElement | 1.72e+05 3.44 % | 1.70e-01 | 1.16e+03 0.04 % | 8.98e+03 0.64 % | 1.01e+04 0.74 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #204: _ZL16LagrangeElement | 1.63e+05 3.44 % | 1.59e-01 | 1.13e+03 0.04 % | 9.06e+03 0.64 % | 1.02e+04 0.74 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #204: _ZL23IntegrateStressFor | 1.37e+05 2.94 % | 1.30e-01 | 1.22e+03 0.04 % | 9.36e+03 0.74 % | 1.06e+04 0.74 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #204: CollectElementPositions | 4.65e+04 1.34 % | 1.17e-02 | 4.60e+01 0.00 % | 4.78e+02 0.00 % | 5.24e+02 0.00 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #204: _ZL13LagrangeNodeInitial | 1.18e+04 0.24 % | 6.52e-02 | 1.90e+02 0.00 % | 1.62e+03 0.14 % | 1.82e+03 0.14 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #292: monitor_main | | | 7.21e+02 0.00 % | | 7.21e+02 0.00 % |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ first touch place | | | | | |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #327: monitor_main | | | | | |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ #479: main | | | | | |
| ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ ↳ loop at LULESH_3D.noGlobal | | | | | |
| ↳ loop at LULESH_3D.noGlobal | | | | | |
| ↳ loop at LULESH_3D.noGlobal | | | | | |
| ↳ LULESH_3D.noGlobal | | | | | |
| ↳ #2164: operator new(unsigned long) | 4.14e+05 8.64 % | 1.73e-01 | 3.05e+03 0.14 % | 2.09e+04 1.54 % | 2.39e+04 1.64 % |
| ↳ #2159: operator new(unsigned long) | 3.76e+05 8.04 % | 1.14e-01 | 4.26e+03 0.14 % | 2.96e+04 2.14 % | 3.39e+04 2.24 % |

[3] Profiling Directed NUMA Optimization on Linux Systems: A Case Study of the Gaussian Computational Chemistry Code

[4] A Tool to Analyze the Performance of Multithreaded Programs on NUMA architectures

NUMPROF

HOW TO KNOW IF WE ARE RIGHT IN A REAL APPLICATION ?

NUMAPROF

10

Sébastien Valat
15/02/2018

- ▶ Take back the idea from **MALT**
 - ▶ **Web interface**
 - ▶ **Source annotation**
 - ▶ **Global metrics**
- ▶ Use intel **Pin**
 - ▶ Permit to **instrument** all **memory accesses**
 - ▶ **Parallel** opposite to valgrind
 - ▶ **Difficulty**: we cannot easily use libs inside the tool
 - ▶ I would have used hwloc and libnuma.....

On access we need...

11

Sébastien Valat
15/02/2018

- ▶ On each access we want to know if it is
 - ▶ **Remote** access
 - ▶ **Local** access
 - ▶ **MCDRAM** access
 - ▶ **Page is pinned**
 - ▶ **Thread is pinned**
- ▶ So, we need to know
 - ▶ Where is **the page**
 - ▶ Where is **the current thread**
- ▶ We can **skip** accesses to **local stack** (overhead 80x -> 40x)

Keep track of page mapping

12

Sébastien Valat
15/02/2018

- ▶ Can **query page location** with

```
int status  
long ret = move_pages(0,1,pages,NULL,&status,0);
```

- ▶ It cost a **system call**
- ▶ **Cannot do** it for **every access**
- ▶ Need to build a **cache**

Similar to kernel page table

13

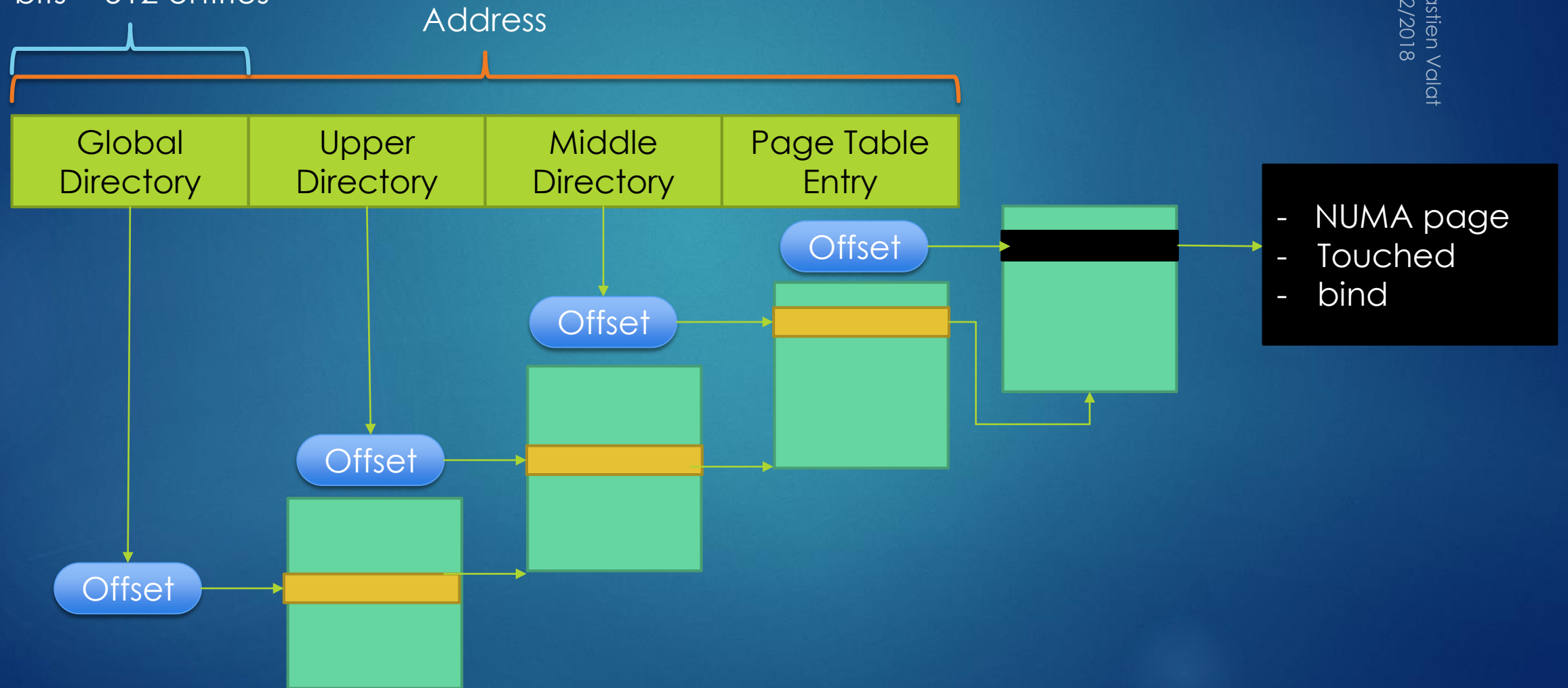
Sébastien Valat
15/02/2018

- ▶ I use the same layout than kernel **page table**
 - ▶ With **multiple levels** of 512 entries
- ▶ For each page **we track**
 - ▶ **NUMA location**
 - ▶ If has **already** been **touched**
 - ▶ If first touch was from the **binded or not binded** thread
- ▶ Need to track **mmap/mremap/munmap**
 - ▶ To **update** page **touched status**

Shadow Page table

14

9 bits = 512 entries

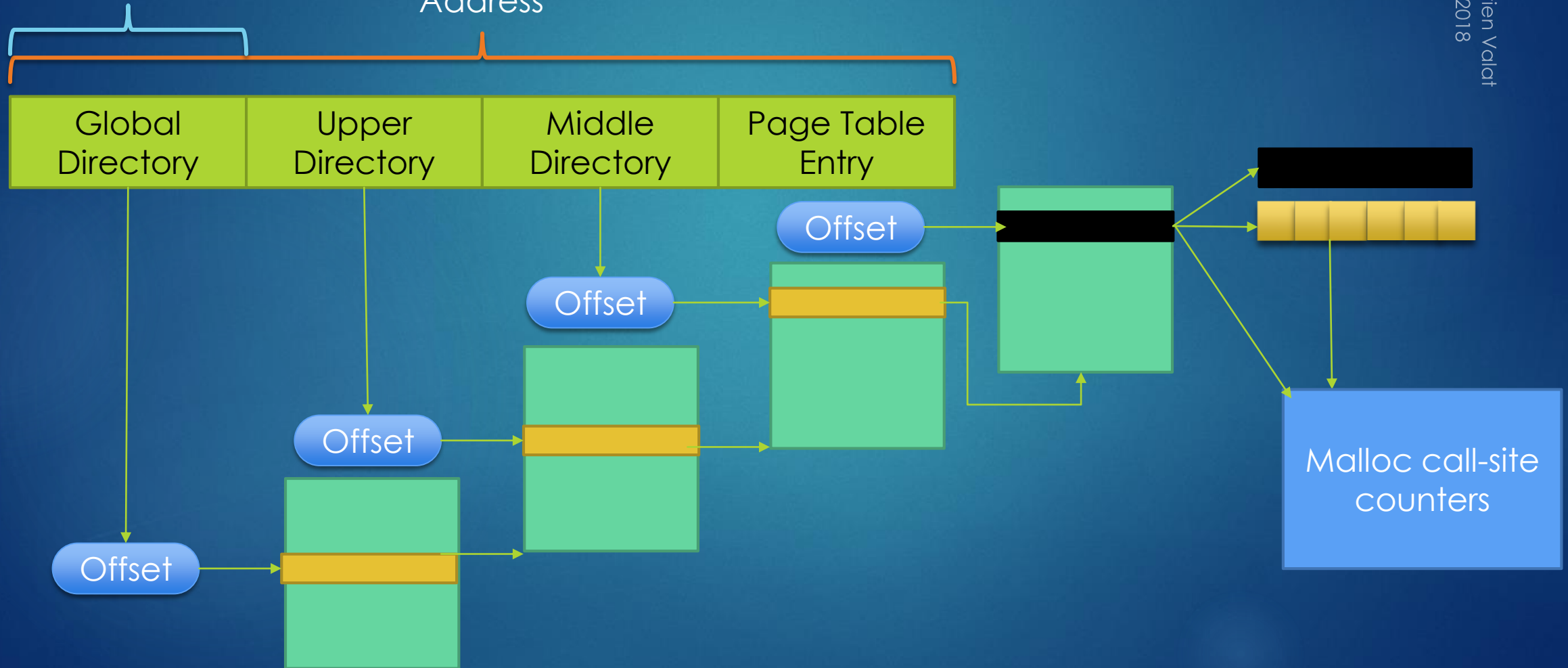


Allocation site

15

9 bits = 512 entries

Address

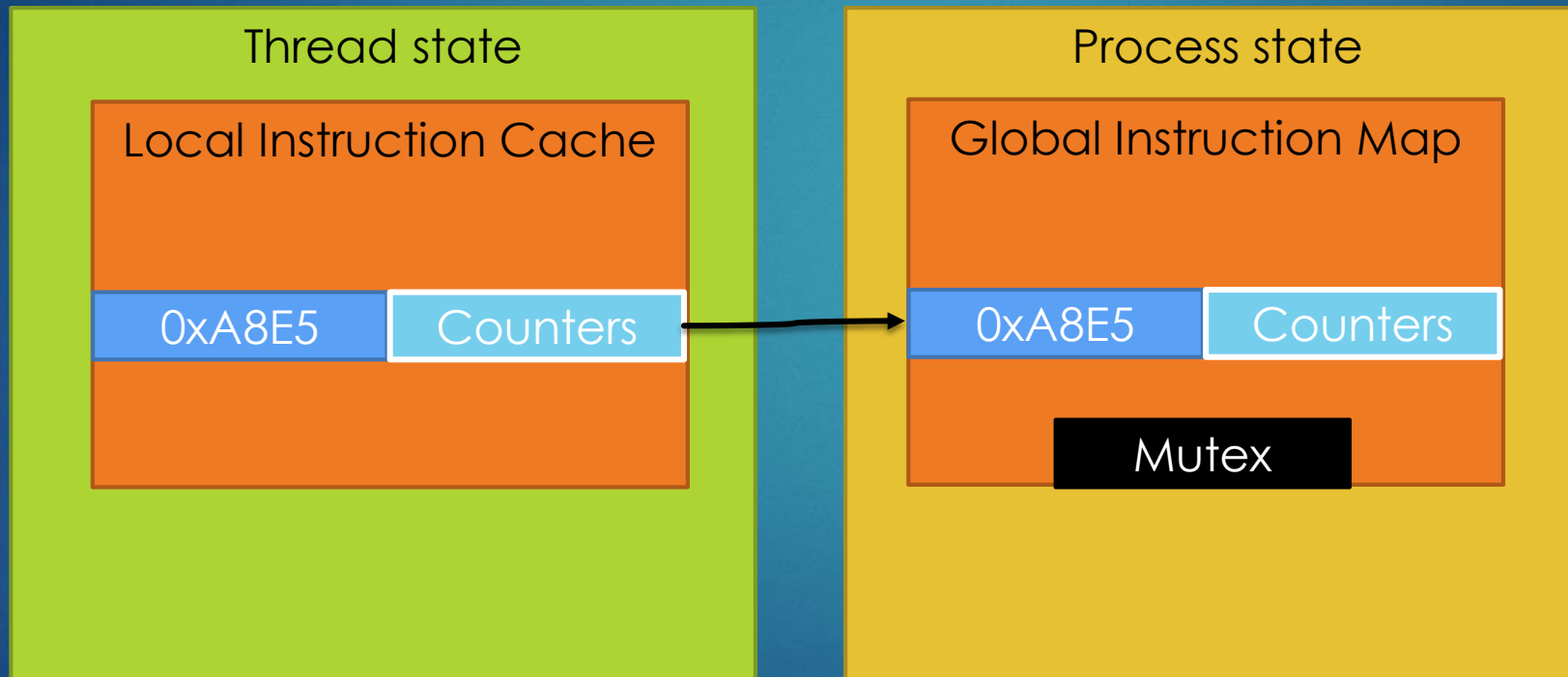


Limit mutexes & atomics

16

Sébastien Valat
15/02/2018

- I use caches to **accumulate locally** and **flush** sometimes

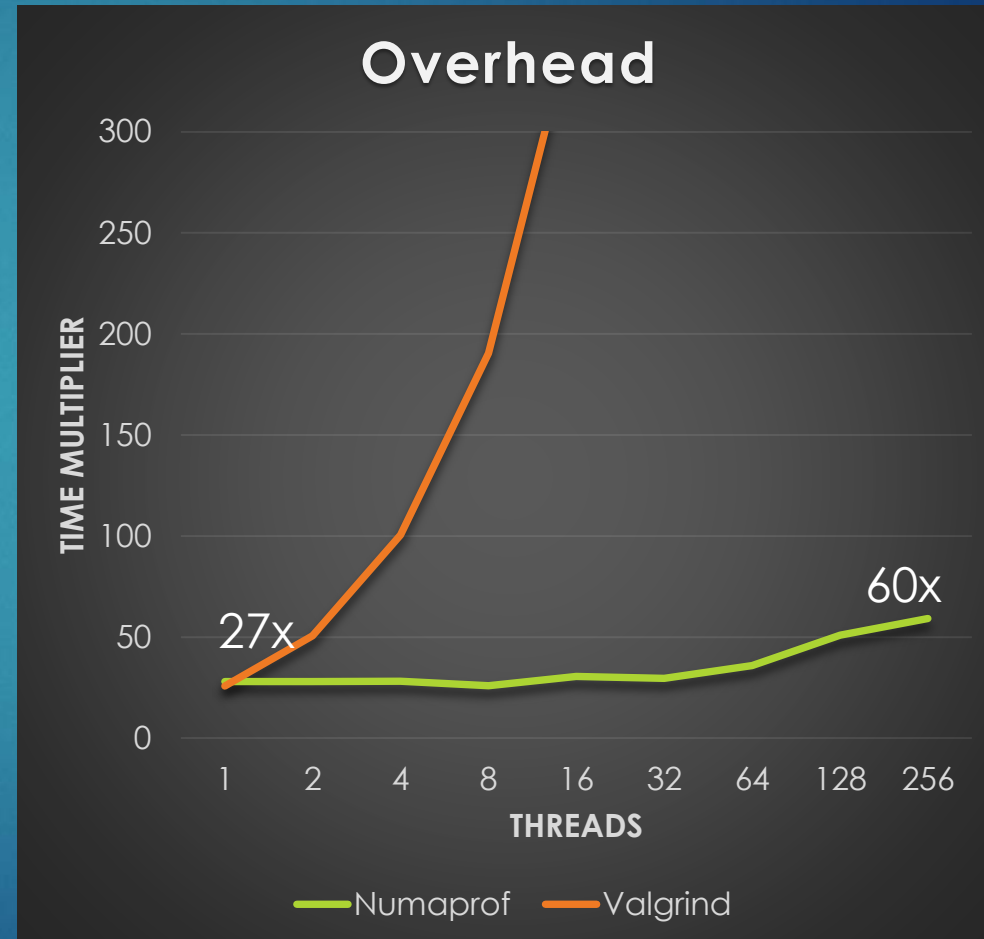


Overhead and scalability

17

Sébastien Valat
15/02/2018

- ▶ Of course overhead is large: **~30x**
- ▶ But is **scale**
- ▶ Example code hydro on **KNL**:



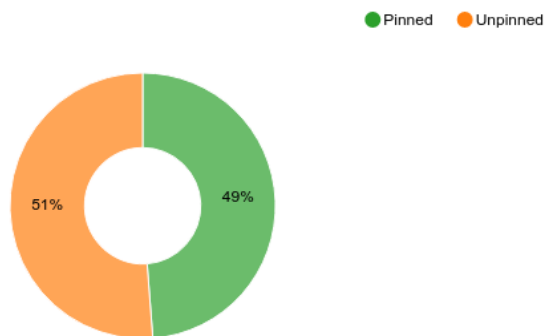
GUI and example

Global summary

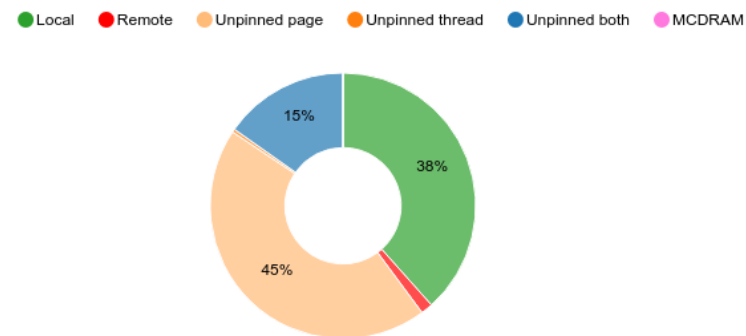
19

Sébastien Valat
15/02/2018

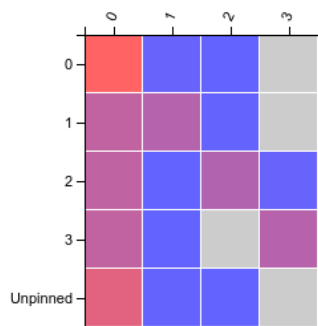
First touch



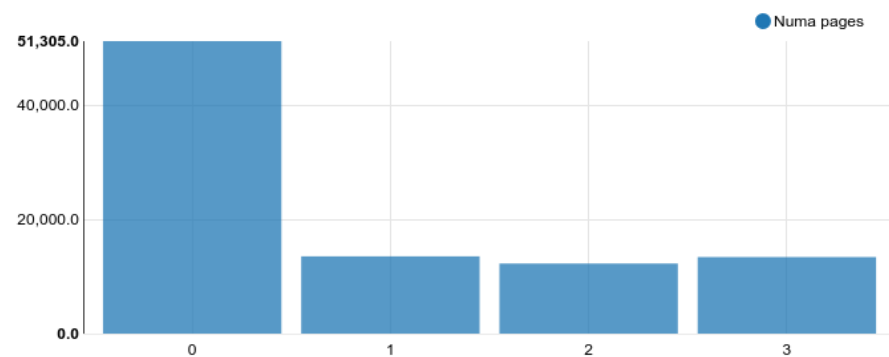
Memory access



Access matrix



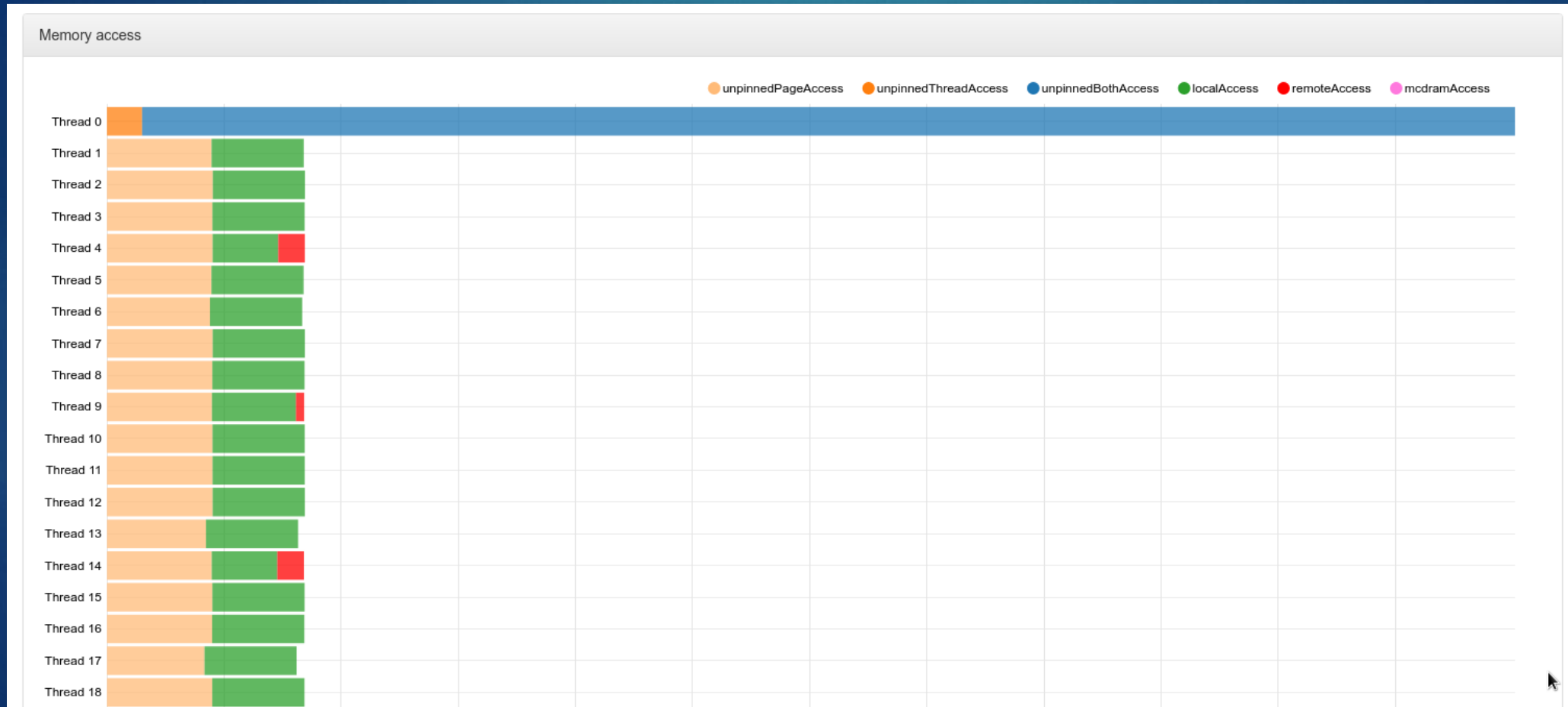
Peak allocated numa pages



Statistics per thread

20

Sébastien Valat
15/02/2018

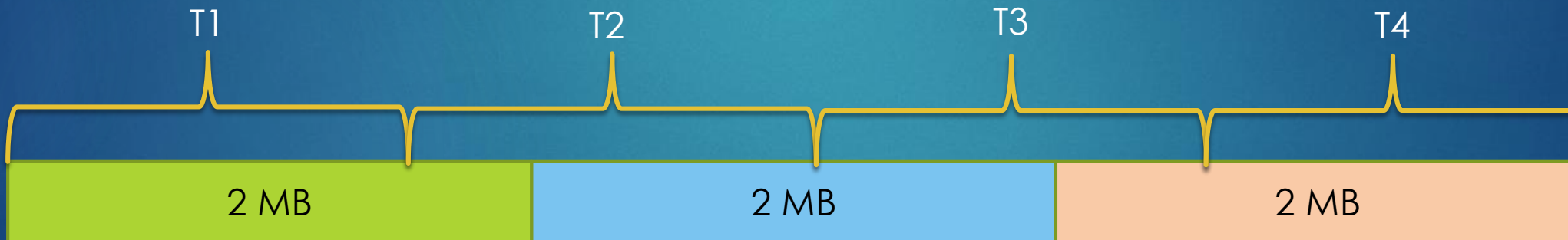


OMP and huge pages

21

Sébastien Valat
15/02/2018

- ▶ Huge pages & thread splitting
- ▶ Most of the time do not match exactly
- ▶ Not a big issue if limited



Details per thread

22

Sébastien Valat
15/02/2018

Numaprof

Home

Threads

Details

Sources

Assembler

Help

←

«

1

2

3

4

5

6

7

8

9

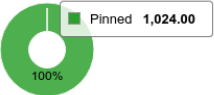

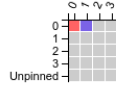
10

»



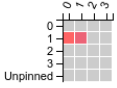
→

5

Thread 4

| | |
|---------------------|---|
| Lifetime | 64.31% → 90.65% |
| CPU thread binding | 4 |
| Numa thread binding | 0 |
| Numa mem. policy | MPOL_DEFAULT on -1 considered as NO_BIND |
| First touch |  |
| Accesses |  |
| Accsses |  |
| Pinning log | <div>At 64.31%, pin thread on node 0</div> <div>At 64.31%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND</div> |

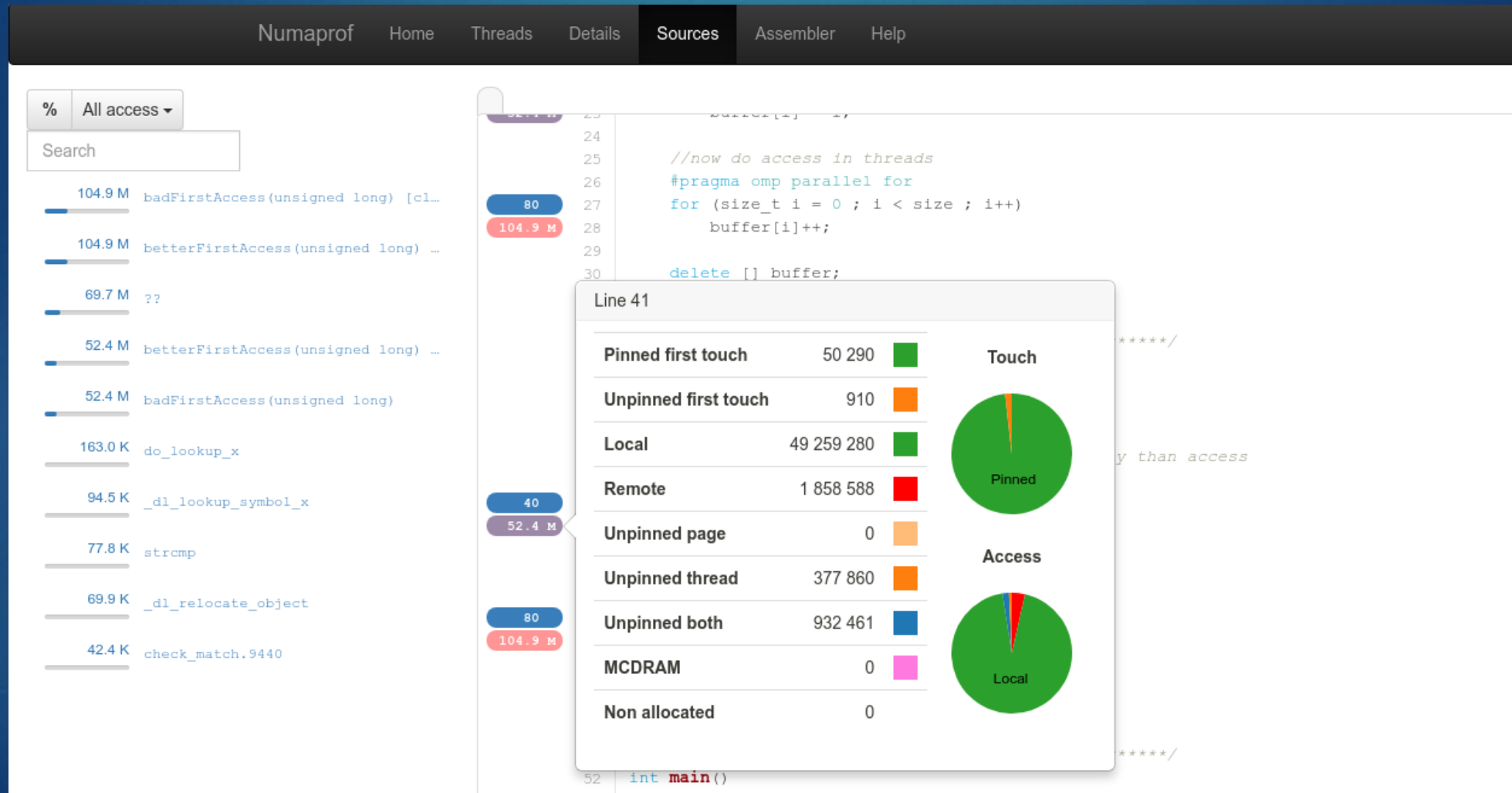
Thread 5

| | |
|---------------------|---|
| Lifetime | 64.32% → 90.91% |
| CPU thread binding | 5 |
| Numa thread binding | 1 |
| Numa mem. policy | MPOL_DEFAULT on -1 considered as NO_BIND |
| First touch |  |
| Accesses |  |
| Accsses |  |
| Pinning log | <div>At 64.32%, pin thread on node 1</div> <div>At 64.32%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND</div> |

Source & asm annotations

23

Sébastien Valat
15/02/2018

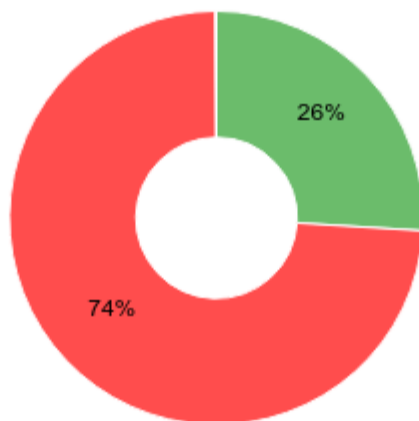


► KNL Without HBM

WITH HBM

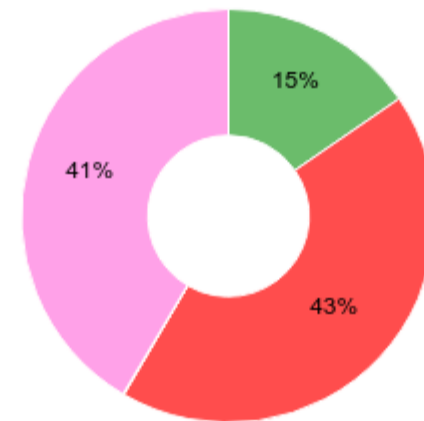
Memory access

Local Remote Unpinned page Unpinned thread
Unpinned both MCDRAM



Memory access

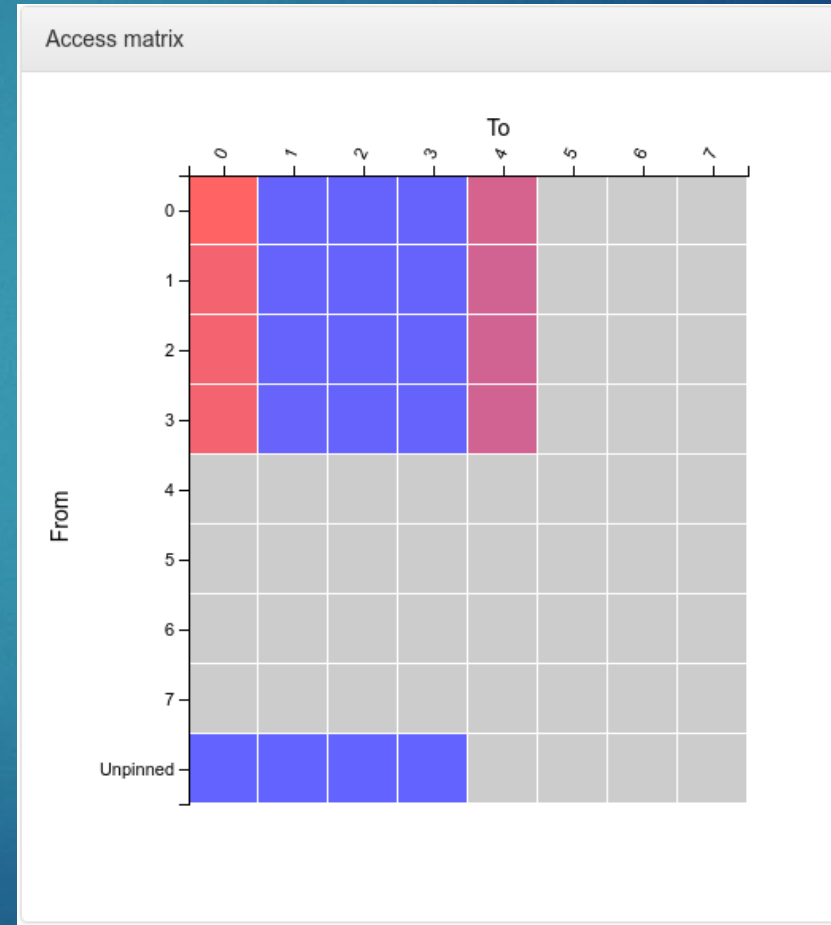
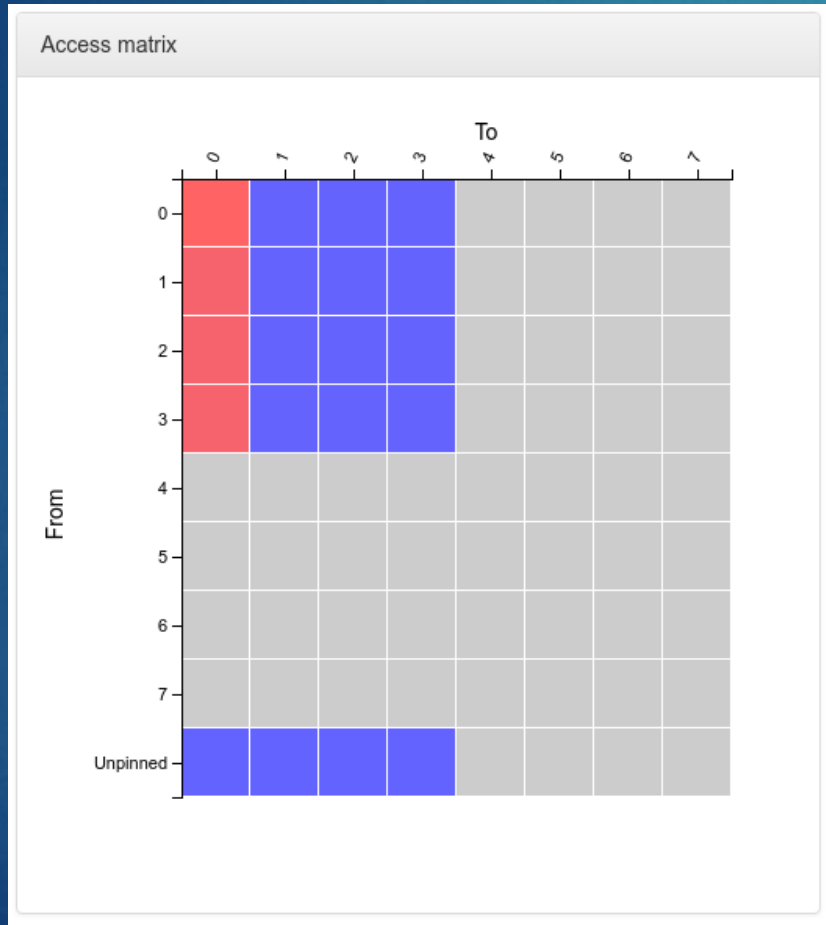
Local Remote Unpinned page Unpinned thread
Unpinned both MCDRAM



Original Hydro access matrix

25

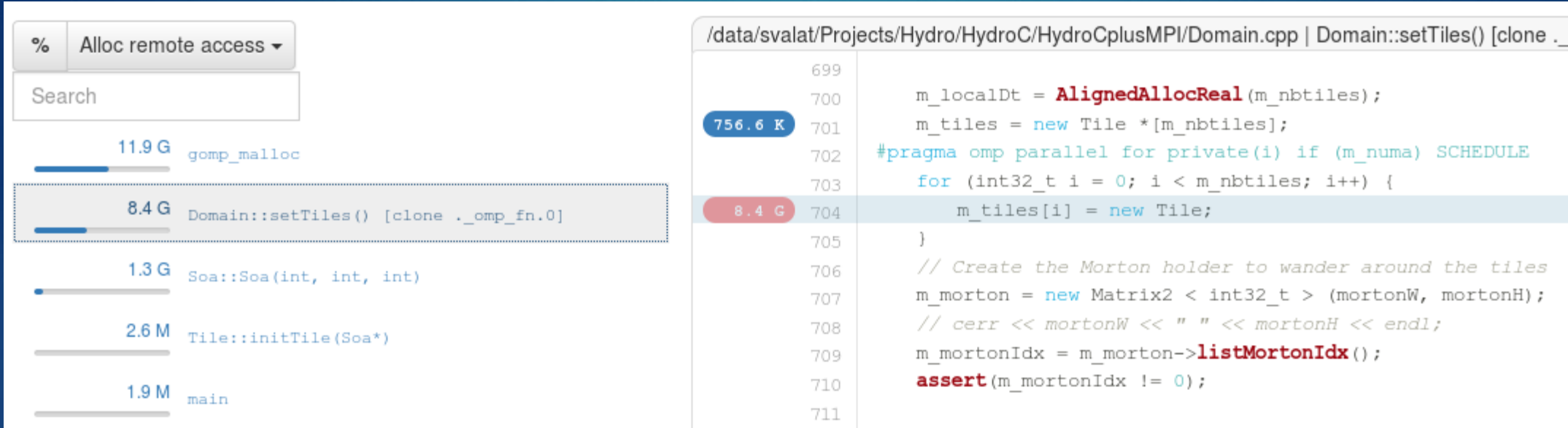
Sébastien Valat
15/02/2018



Ordering issue

26

Sébastien Valat
15/02/2018

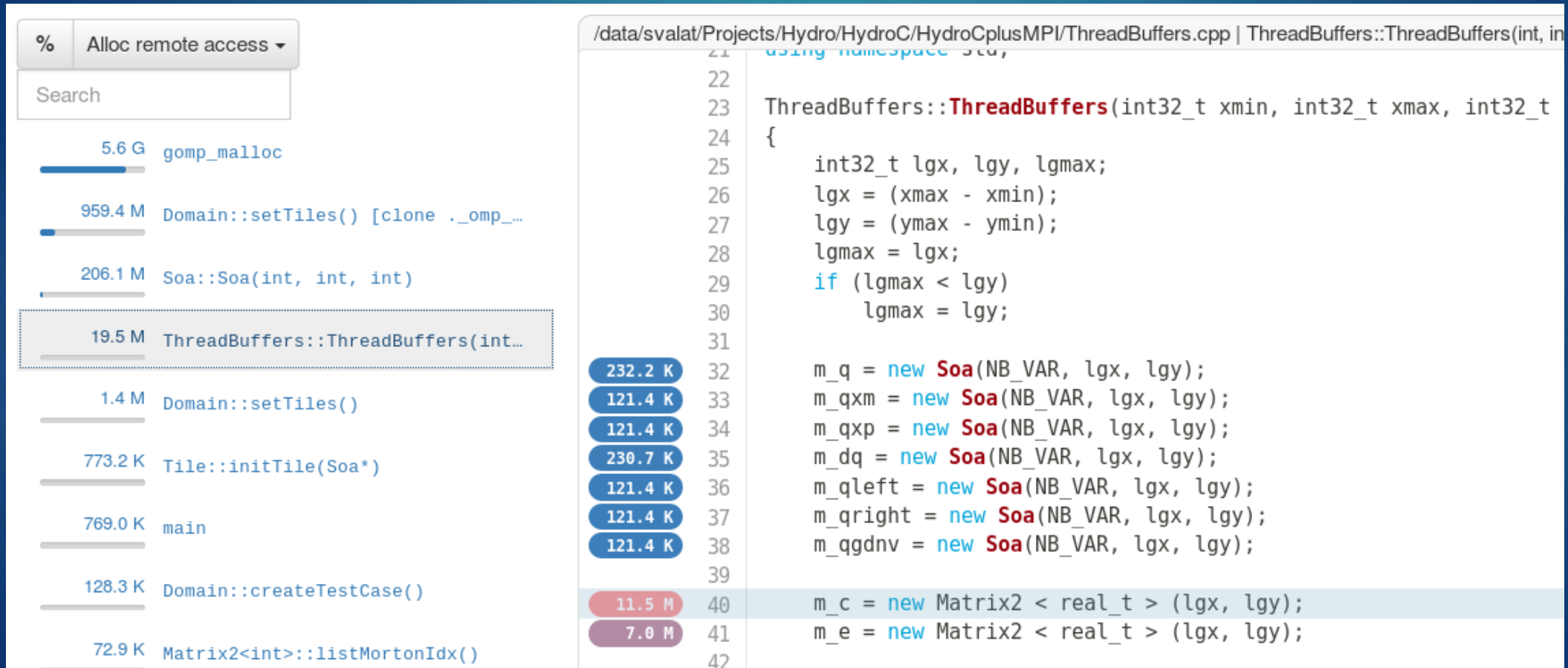


```
#pragma omp parallel for private(i) if (m_numa) SCHEDULE
for (int32_t i = 0; i < m_nbtiles; i++) {
    int t = m_mortonIdx[i];
    m_tiles[t] = new Tile;
}
```

Non parallel allocations

27

Sébastien Valat
15/02/2018



Parallel allocations

28

Sébastien Valat
15/02/2018

► Original

```
for (int32_t i = 0; i < m_numThreads; i++) {  
    m_buffers[i] = new ThreadBuffers(...);  
    assert(m_buffers[i] != 0);  
}
```

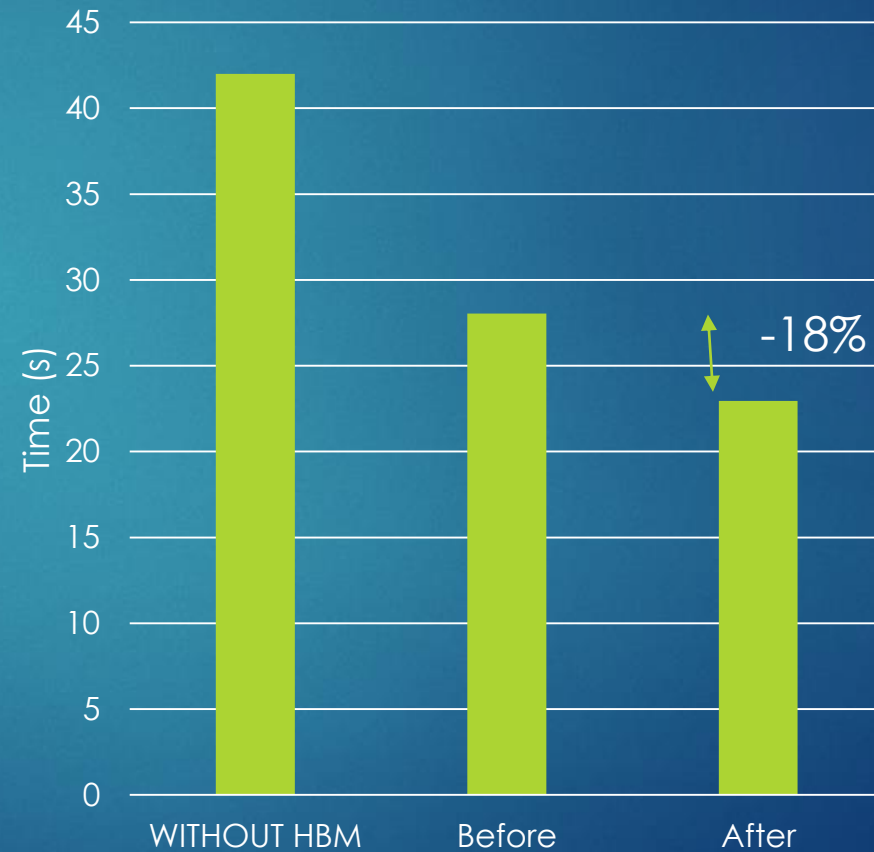
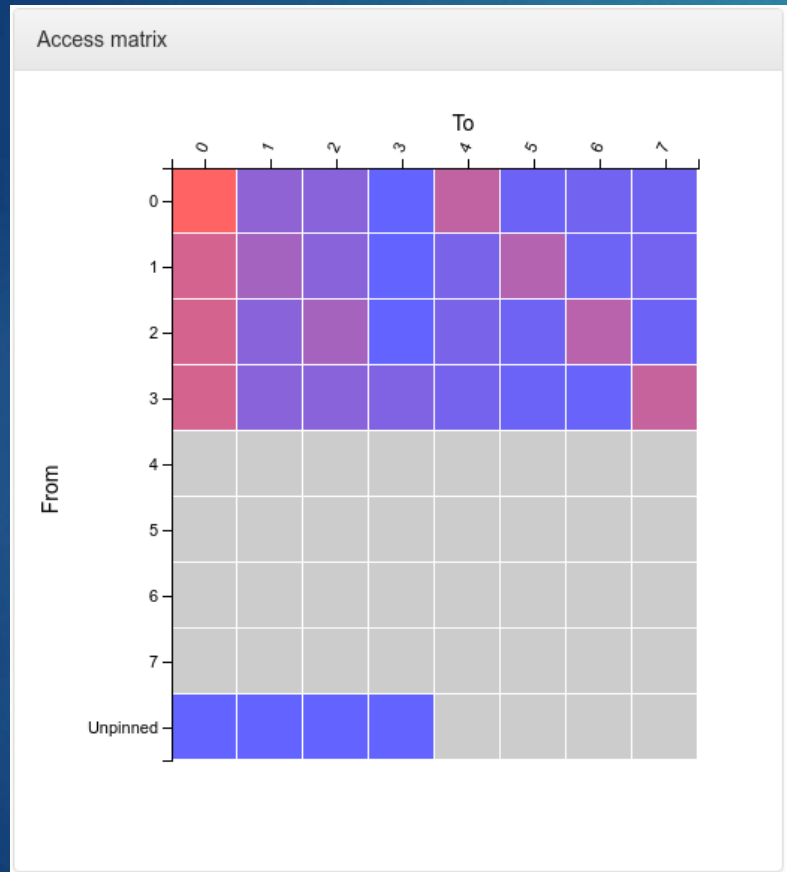
► Modified

```
#pragma omp parallel  
{  
    int i = omp_get_thread_num();  
    #pragma omp critical  
    m_buffers[i] = new ThreadBuffers(..);  
    assert(m_buffers[i] != 0);  
}
```


Speed up obtained on Hydro

29

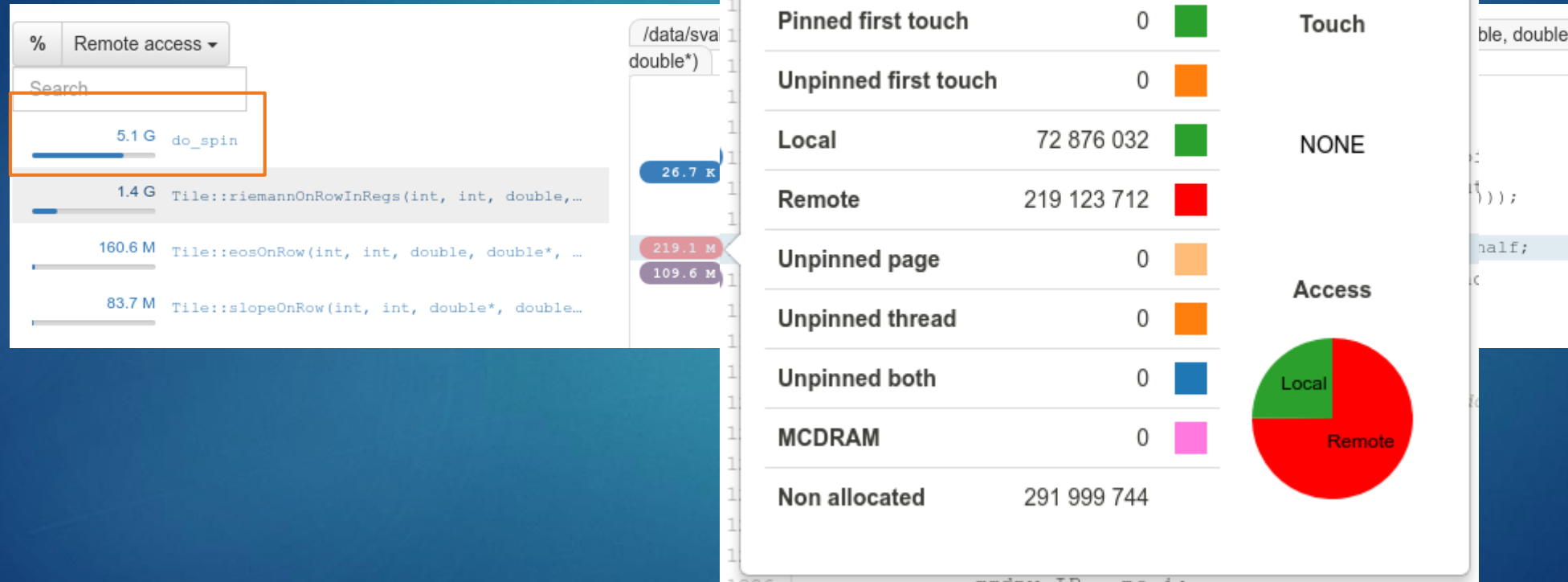
Sébastien Valat
15/02/2018



Remaining consts

30

Sébastien Valat
15/02/2018



Conclusion

31

Sébastien Valat
15/02/2018

- ▶ Tool easy to use coming with a nice GUI
- ▶ Useful to check an application.

- ▶ Of course, still a lot of works to do

- ▶ Add support of **call stacks**
- ▶ Consider **cache simulation**
- ▶ **Optimizations**
- ▶ **Time charts**

<http://memtt.github.io/>

- ▶ Also want to support **DynamorRIO** and **valgrind**.

BACKUP

```
hwloc-bind --cpubind node:0 ./mycommand
```

```
hwloc-bind --membind node:0 ./mycommand
```

```
numactl --membind -m 0 ./mycommand
```

HOW TO CONTROL MEMORY placement

FIRST SOLUTION : BIND THE PROCESS AT LAUNCH TIME

FINE TO SELECT **MCDRAM** IN **SNC1** MODE

BUT WHAT IF WE HAVE **4 NUMA** NODES ?

NEED TO CONTROL AT **THREAD LEVEL**


```
cpu_set_t cpuset;  
CPU_ZERO(&cpuset);  
CPU_SET(1,&cpuset);  
  
pthread_set th = pthread_self();  
pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
```

Thread binding

BIND A THREAD TO A GIVEN CORE

```
KMP_AFFINITY=scatter ./mycommand
```

```
OMP_PROC_BIND=TRUE ./mycommand
```

Extracted metrics

35

Sébastien Valat
15/02/2018

- ▶ First touch
 - ▶ **Pinned** first touch
 - ▶ **Unpinned** first touch
- ▶ Accesses
 - ▶ **Local** access
 - ▶ **Remote** access
 - ▶ **Unpinned thread** access
 - ▶ **Unpinned page** access
 - ▶ **Unpinned both** access
 - ▶ **MCDRAM** access



EXAMPLE OF RUN ON 8 THREADS, 2 NUMA NODES

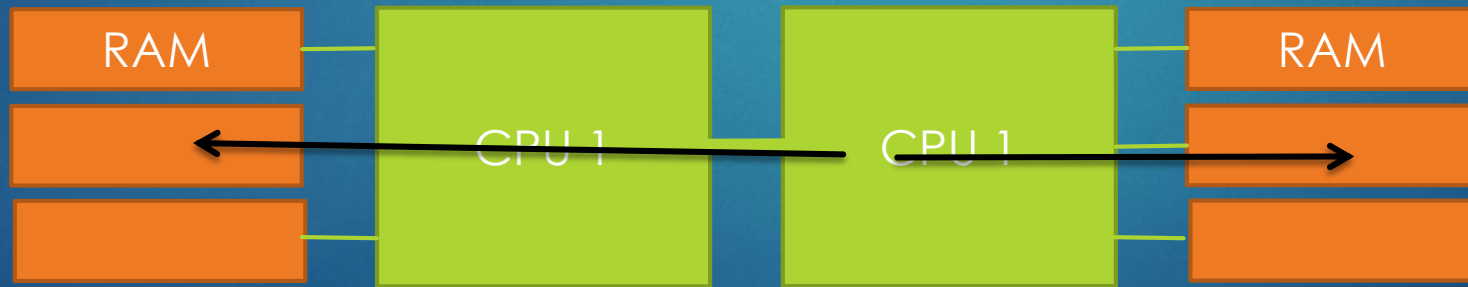
THIS IS TRUE ONLY IF WE BIND THE THREADS

What is NUMA ?

37

Sébastien Valat
15/02/2018

- ▶ Each CPU has its **own memory**
- ▶ Access to **remote memory** we need to **go through the owner CPU**



Want to link to allocation site

38

Sébastien Valat
15/02/2018

- ▶ I want to provide statistics on **allocation site**
- ▶ Need on **each access** to know **where** the bloc **was allocated**
- ▶ Add **entries** into the **page table**
- ▶ **Split** page into blocs of **8 bytes**
- ▶ **Store** a **pointer** for each bloc to point the **segment descriptor**
- ▶ **Issue** if allocation are **smaller** then 8 bytes (**Incompatible** with jemalloc)