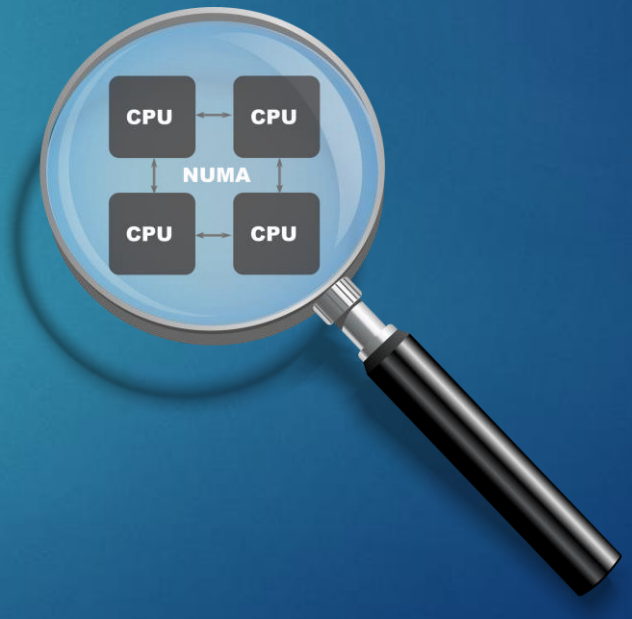
Sébastien Valat - COLOC - 2018
27/08/2018

NUMAPROF

A NUMA MEMORY PROFILER

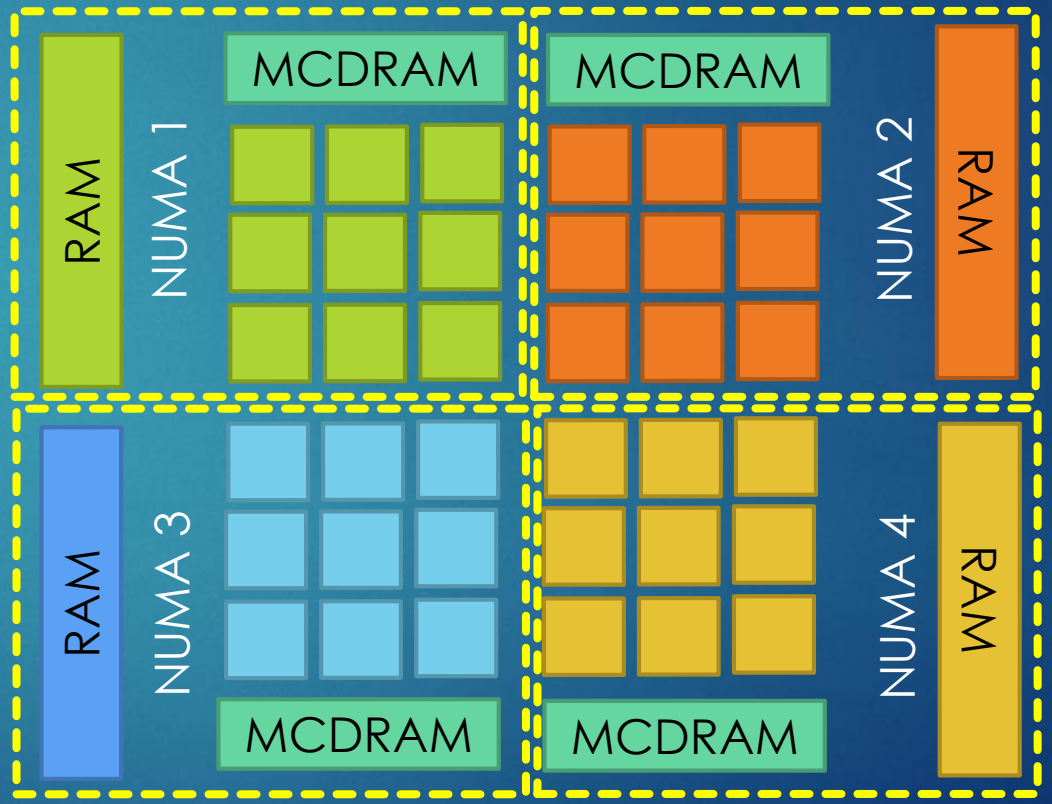


PLAN

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

Today topology

- **NUMA** is now **common** in **HPC**
- Even inside the CPU : **Intel KNL**
- Extra memory (**MCDRAM)**
- Memory **location** is **implicit** on **first touch**

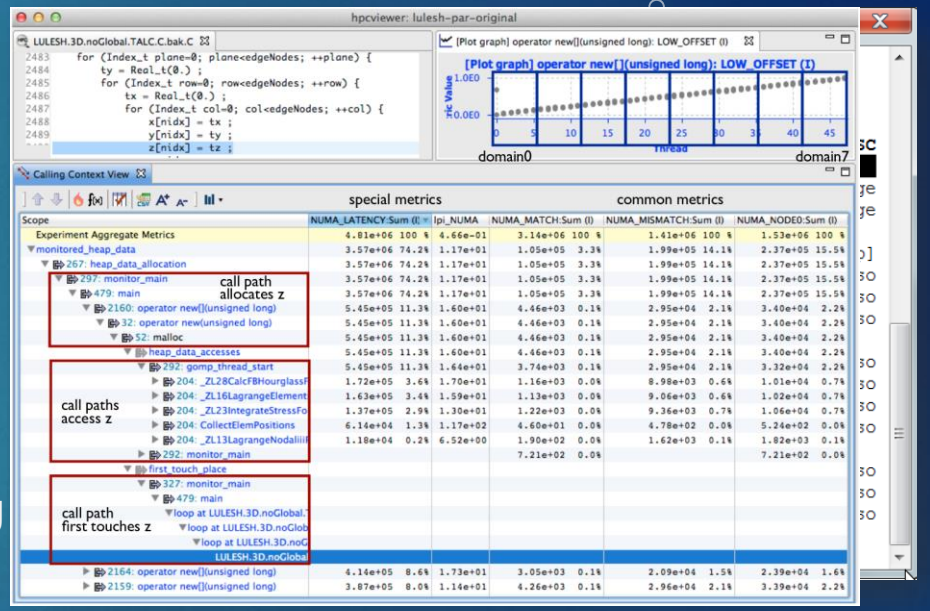


Wish list for a profiling tool…

- Do we make **remote accesses**
- Need to know **where**…
- Know **which allocation contain issues**
- Know **where** the **first touch** has been done
- On KNL we want to check **MCRAM accesses**

Existing tools

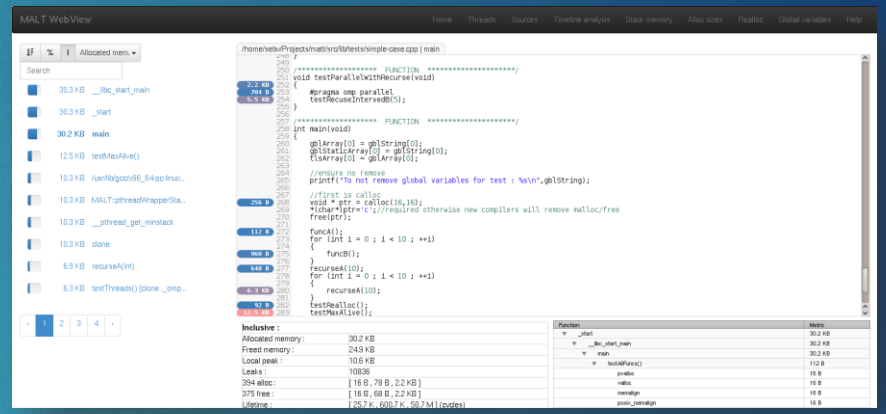
- **MemProf** [1], a research paper from Grenoble
  - Hardware feature provided by AMD & kernel module, No GUI
- **SNPERF** [2], again a research project
  - Link utilization on time chart
- **NUMAgrind** [3]
  - Looks nice but not available
- **Numatop**
  - Similar to top, global profiling
- **HPCToolkit** [4]
  - Also have a nice interface, but hw counters & sampling
- **Tabarnac**
  - Based on pintool like us but static html files.



[1] Renaud Lachaize and Al. MemProf: A Memory Profiler for NUMA Multicore Systems.
[2] U. Prestor and Al., "An application-centric ccNUMA memory profiler,"
[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

NUMAPROF

- 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

```
numaprof ./my_prog –my-prog-option
numaprof-webview ./numaprof-my_prog-10088.json
numaprof-qt ./numaprof-my_prog-10088.json
```

How it works

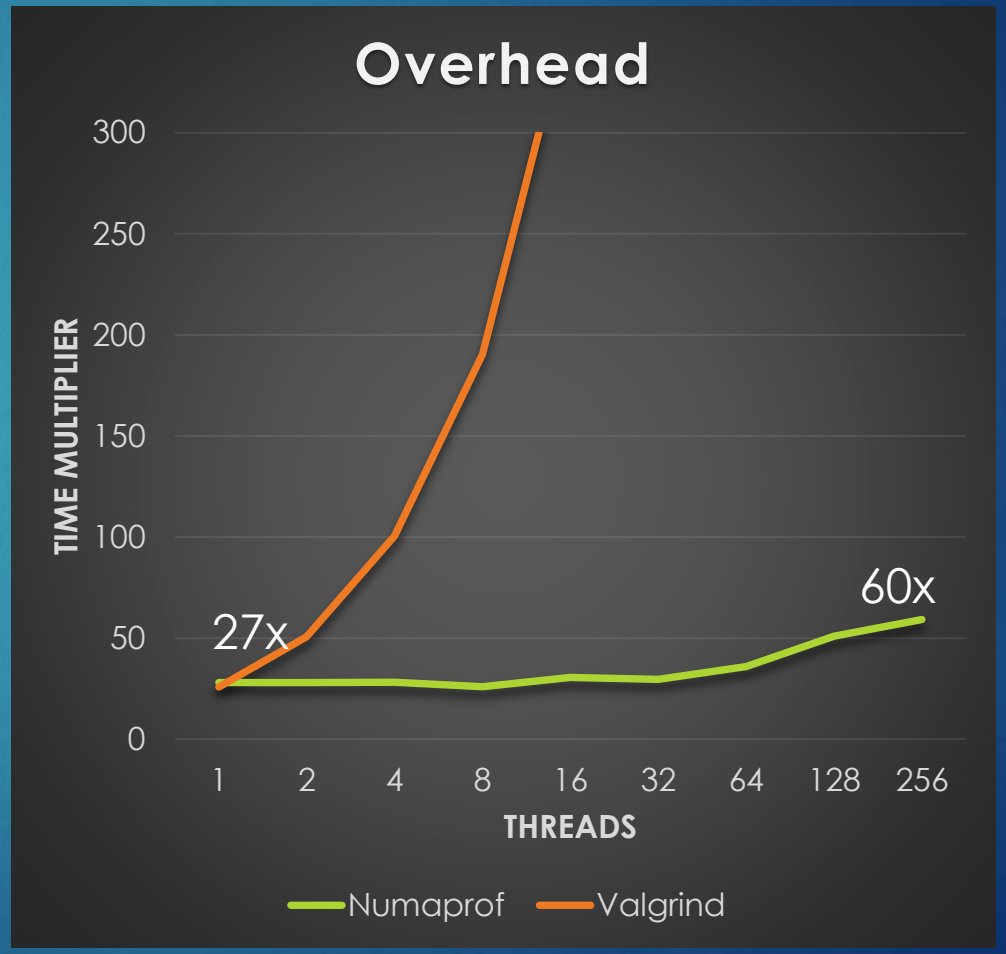
- Pin **instrument** all **Load / Store** (similar to valgrind)
- Track the **memory mapping state** (mmap/munmap/mremap)
- Track **theads location**
- On memory access, we check
  - Location of the **page**
  - Location of the **thread**
- We can **skip** accesses to **local stack** (overhead divided by 2)

On access we need…

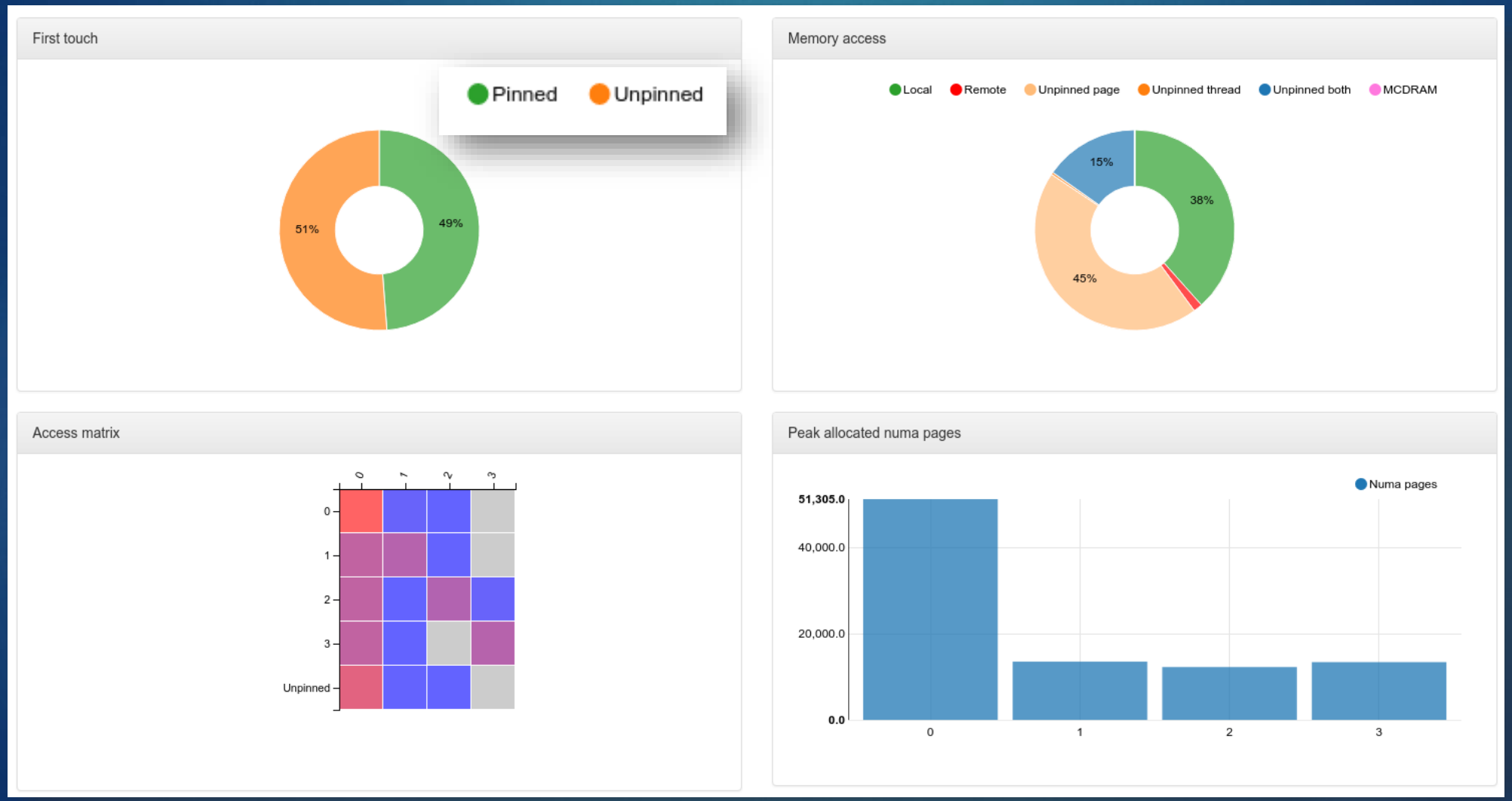
- On each access we want to know if it is
  - **Remote** access
  - **Local** access
  - **MCDRAM remote of local** access
  - **Page is pinned**
  - **Thread is pinned**
- Report on
  - **Access site**
  - **Allocation site**
  - **First touch site**

Overhead and scalability

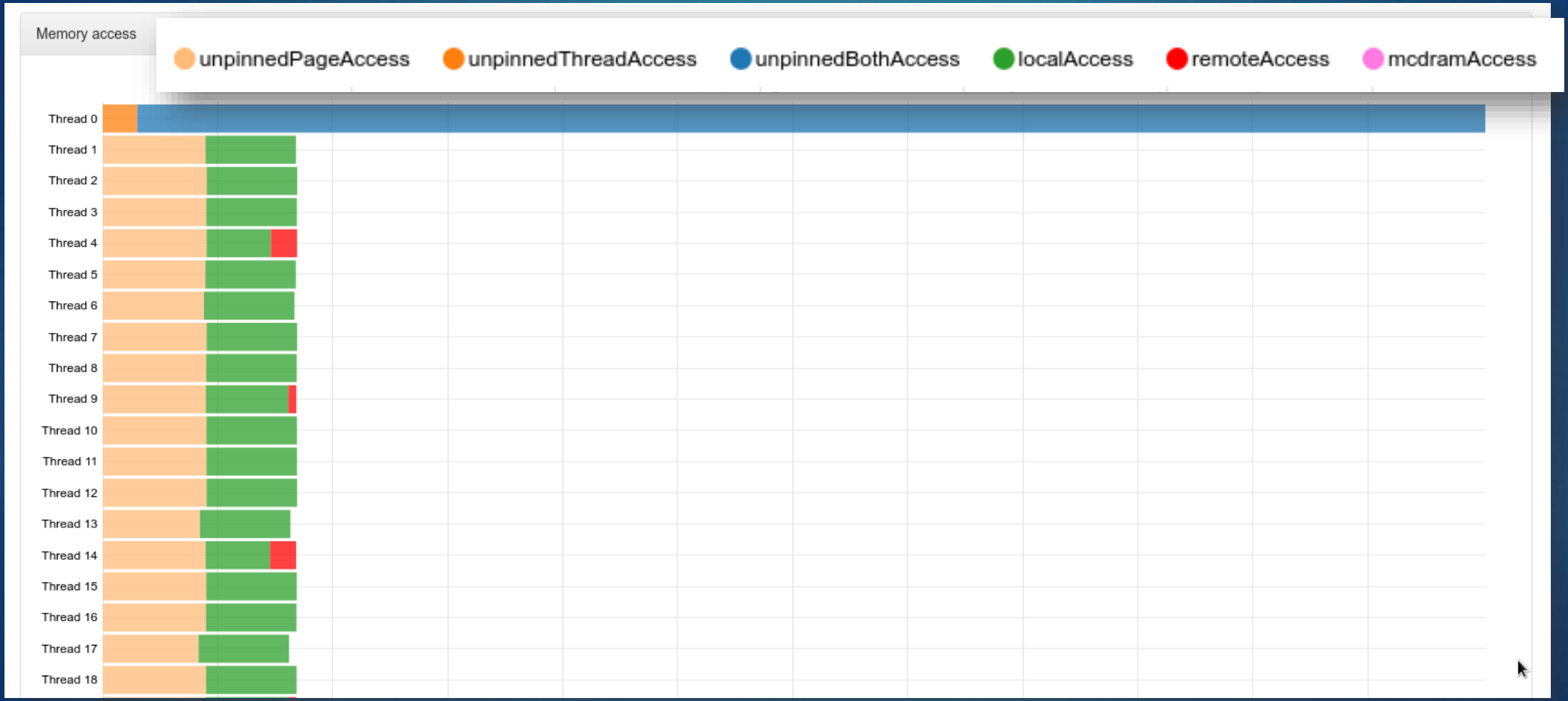
- Of course overhead is large: **~30x**
- But is **scale**
- Example code HydroC on **KNL**:
- We **can** reduce a **factor 2** by **packing** the **load/store** tracking **but loose precision**



Global summary



Statistics per thread



Source & asm annotations

Numaprof | Home | Threads | Details | Sources | Assembler | Help

% | All access

Search

104.9 M	badFirstAccess(unsigned long) [cl…
104.9 M	betterFirstAccess(unsigned long) …
69.7 M	??
52.4 M	betterFirstAccess(unsigned long) …
52.4 M	badFirstAccess(unsigned long)
163.0 K	do_lookup_x
94.5 K	_dl_lookup_symbol_x
77.8 K	strcmp
69.9 K	_dl_relocate_object
42.4 K	check_match.9440

```
24
25      //now do access in threads
26      #pragma omp parallel for
27      for (size_t i = 0 ; i < size ; i++)
28          buffer[i]++;
29
30      delete [] buffer;
```

80 / 104.9 M

Line 41

Pinned first touch	50 290
Unpinned first touch	910
Local	49 259 280
Remote	1 858 588
Unpinned page	0
Unpinned thread	377 860
Unpinned both	932 461
MCDRAM	0
Non allocated	0

Touch

Access


```
*****/
y than access
*****/
int main()
```

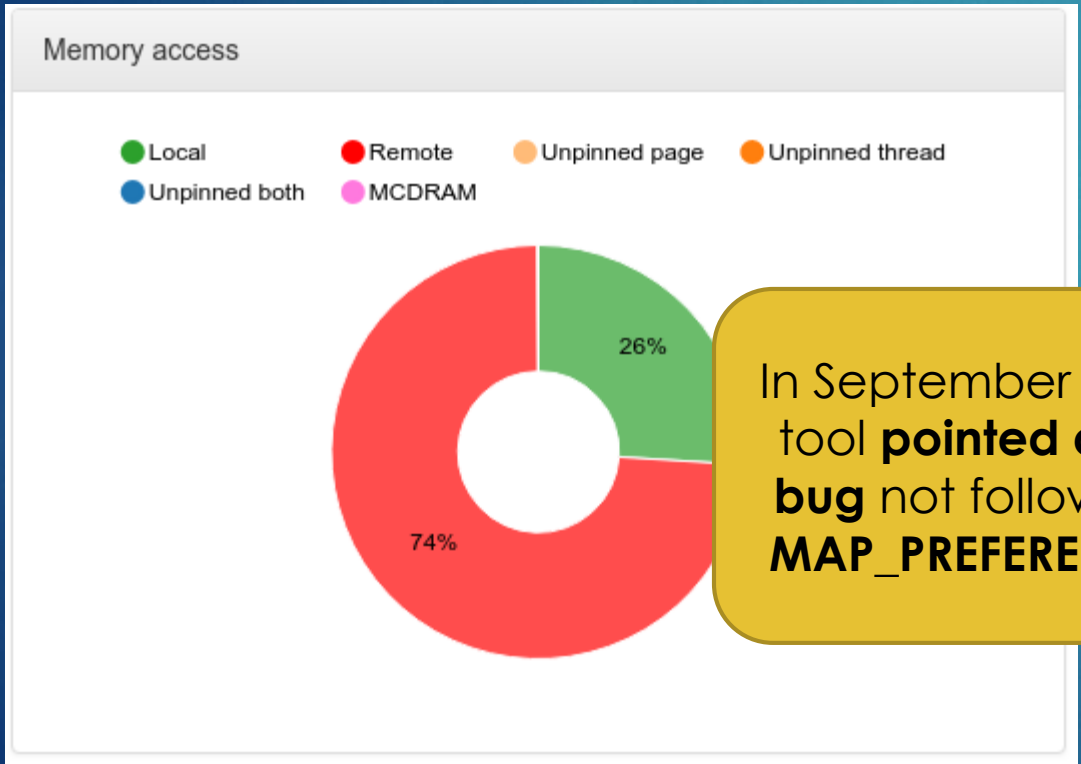
40 / 52.4 M

80 / 104.9 M

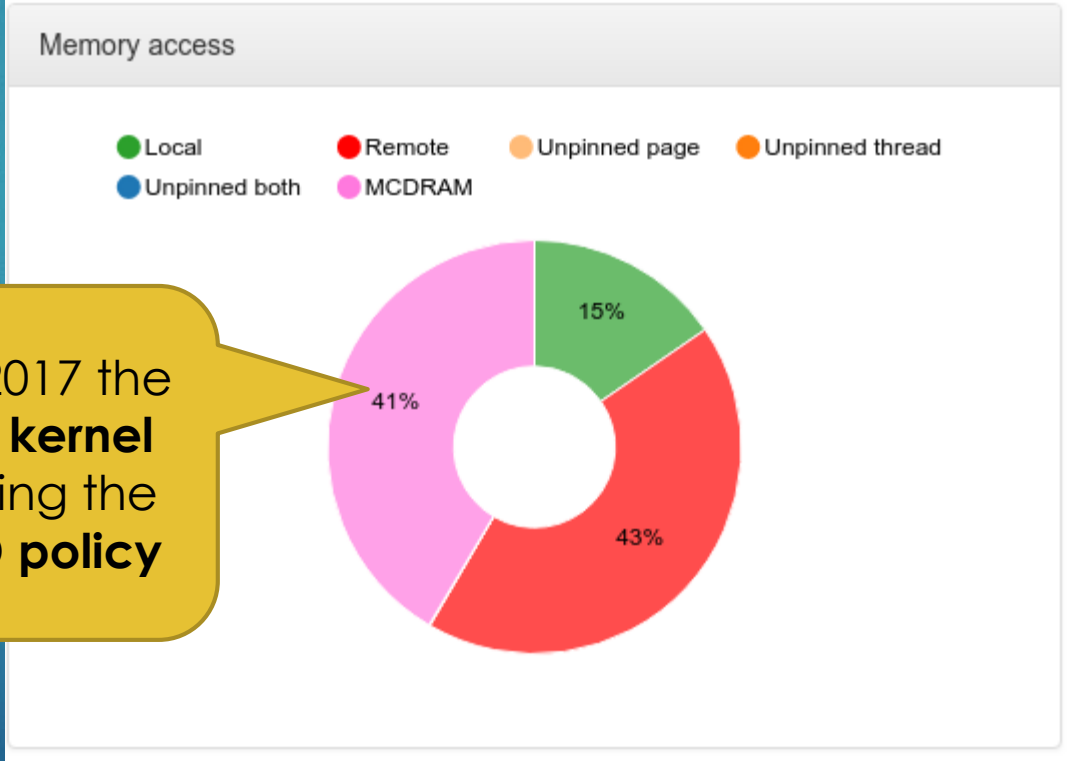
Code Hydro no Intel KNL

Sébastien Valat
27/08/2018

- KNL Without MCDRAM



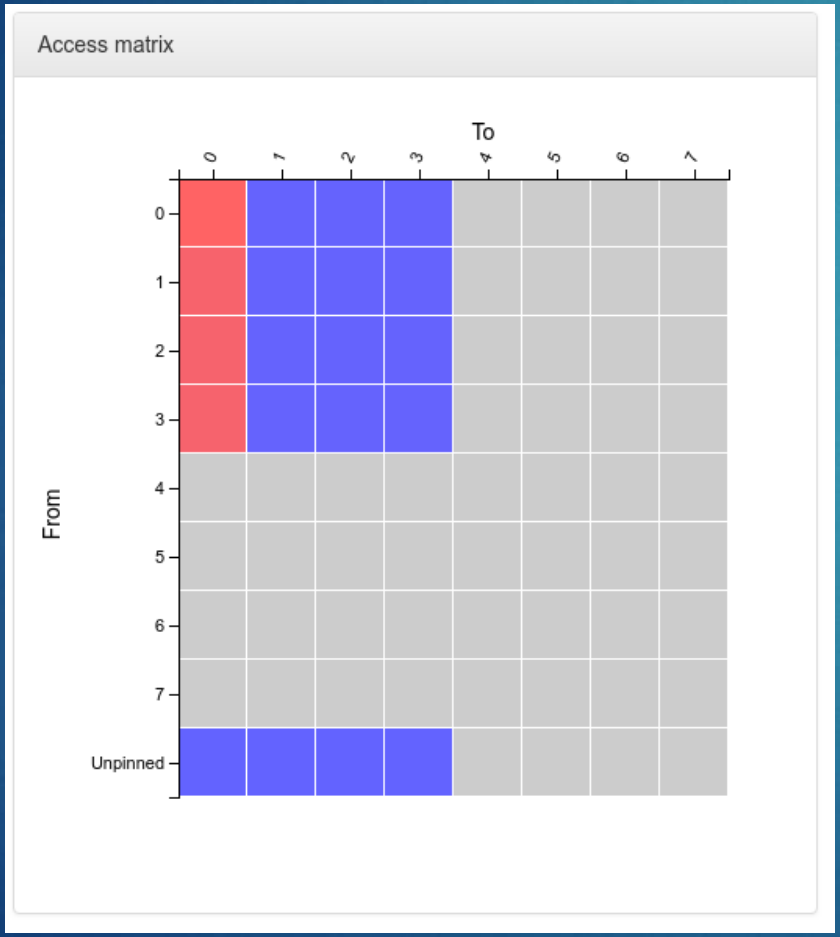
WITH MCDRAM



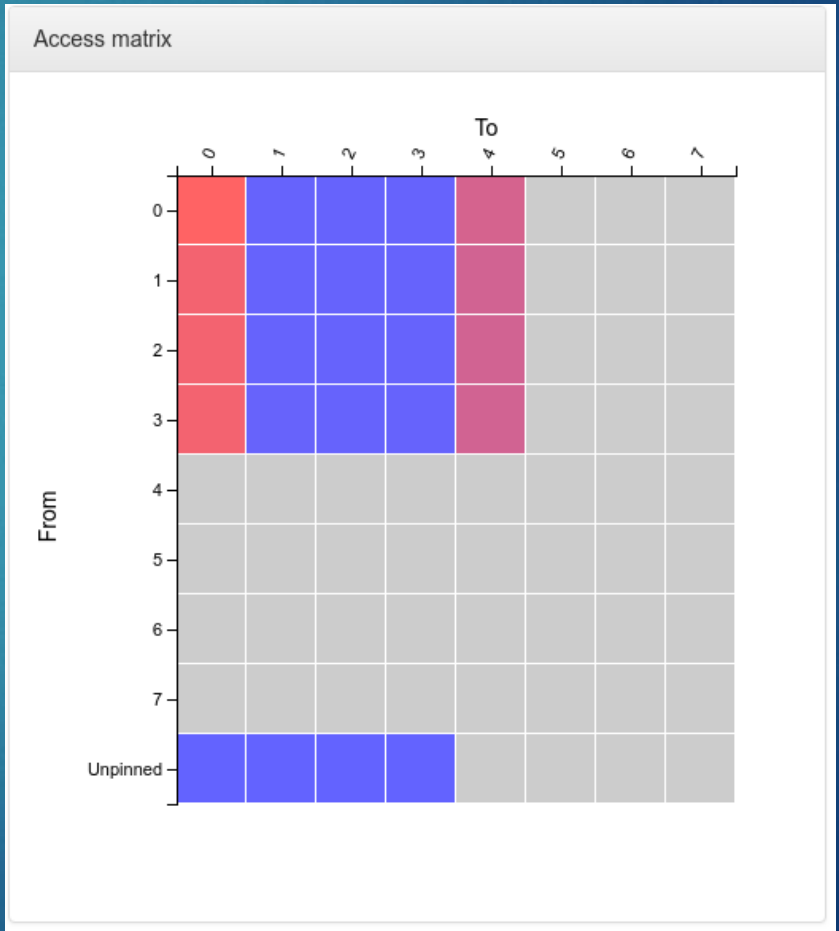
In September 2017 the tool **pointed a kernel bug** not following the **MAP_PREFERED policy**

Original Hydro access matrix

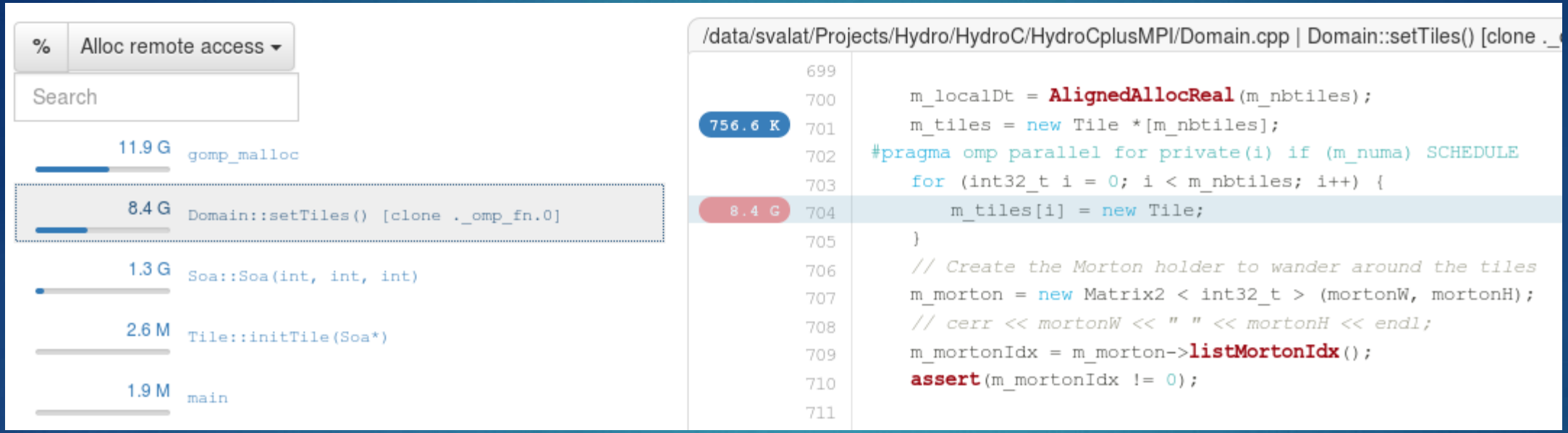
- KNL Without MCDRAM



WITH MCDRAM

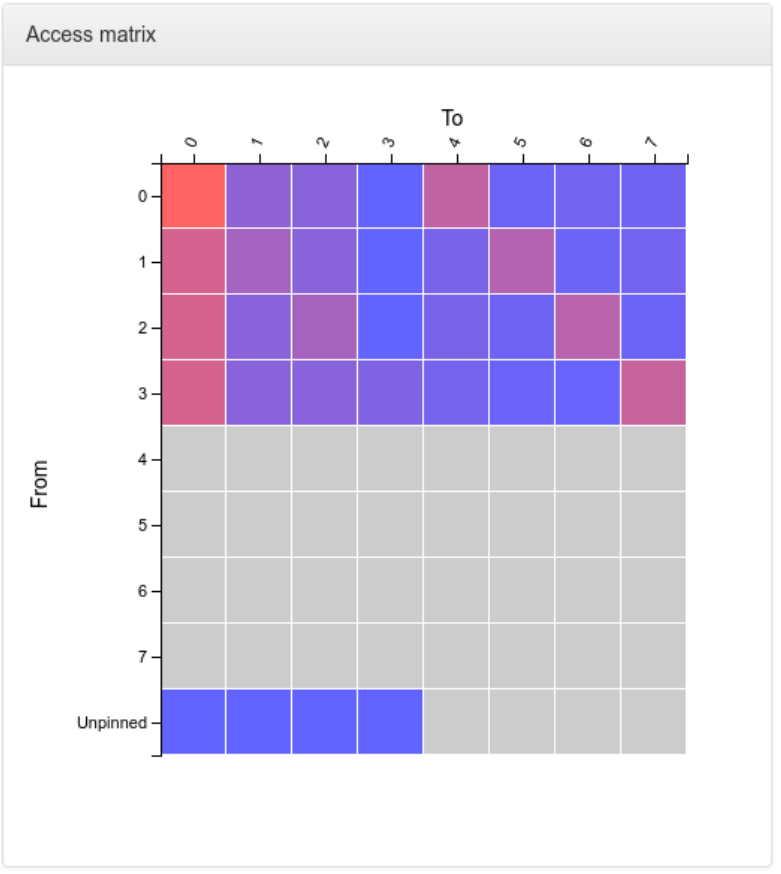


Ordering issue




```
#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;
        }
```

Speed up obtained on Hydro



Conclusion

- 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**
- Also wan to to support **DynamorRIO** and **valgrind**.

Open Source v1.0 on
http://memtt.github.io/

BACKUP

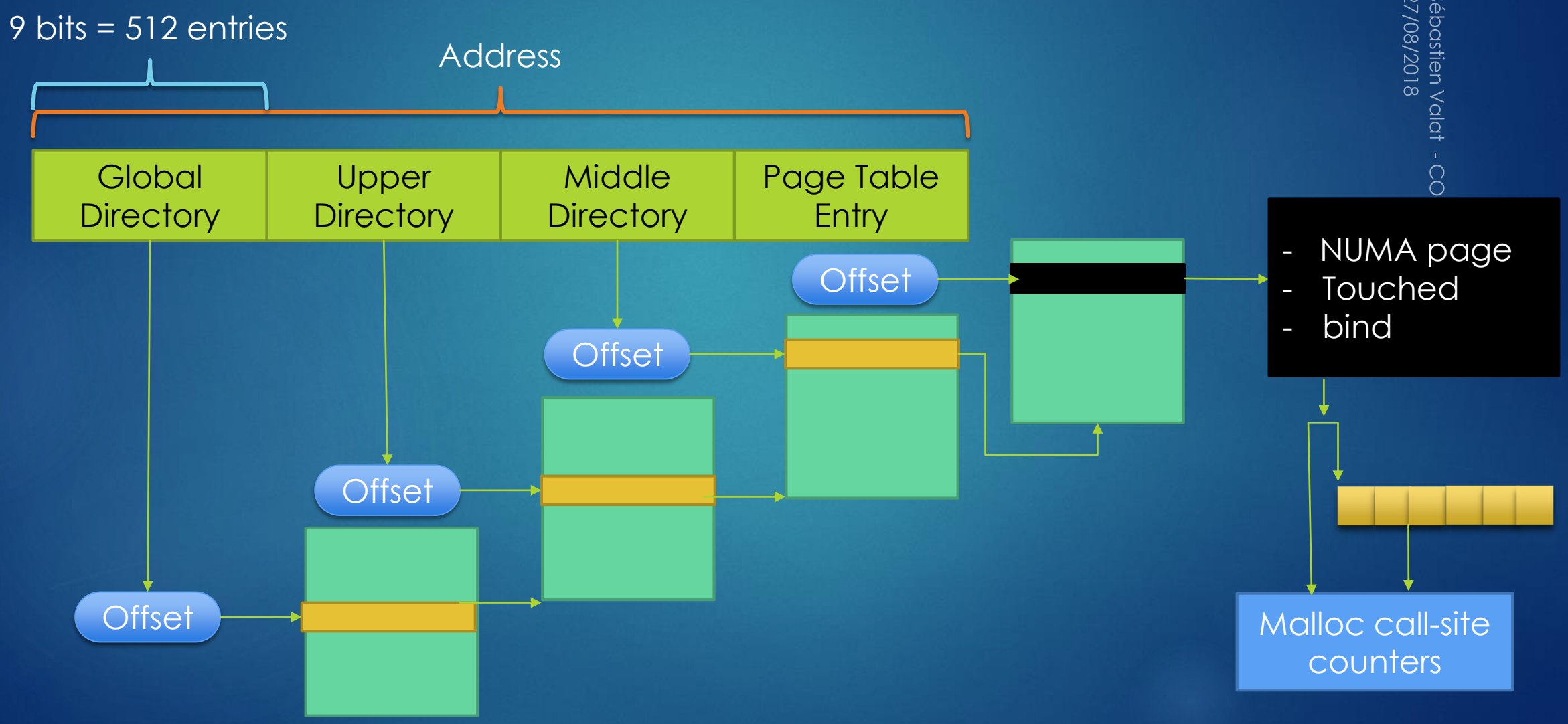
Keep track of page mapping

- Can **query page location** with

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

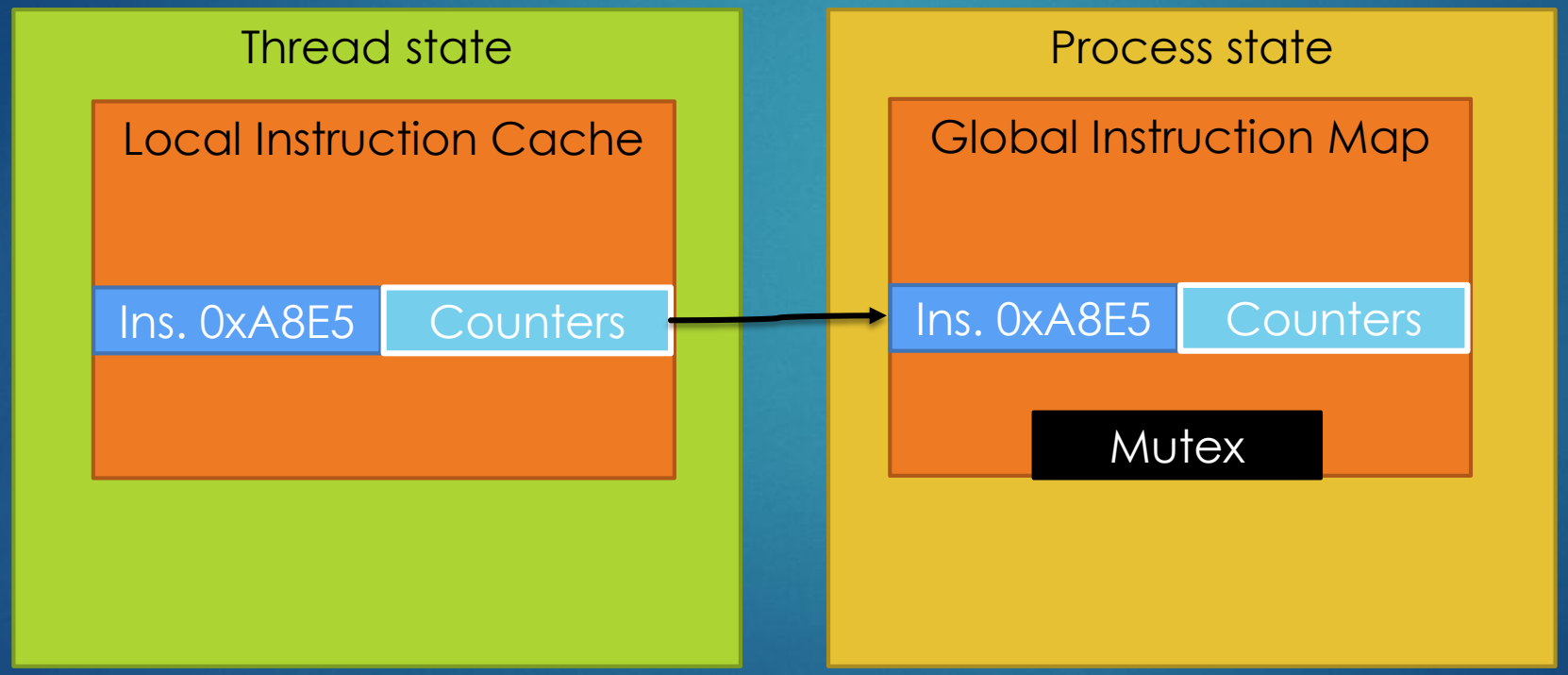
- It cost a **system call**
- **Cannot do** it for **every access**
- Need to build **a cache**

Shadow Page table



Limit mutexes & atomics

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



Non parallel allocations

% | Alloc remote access

Search

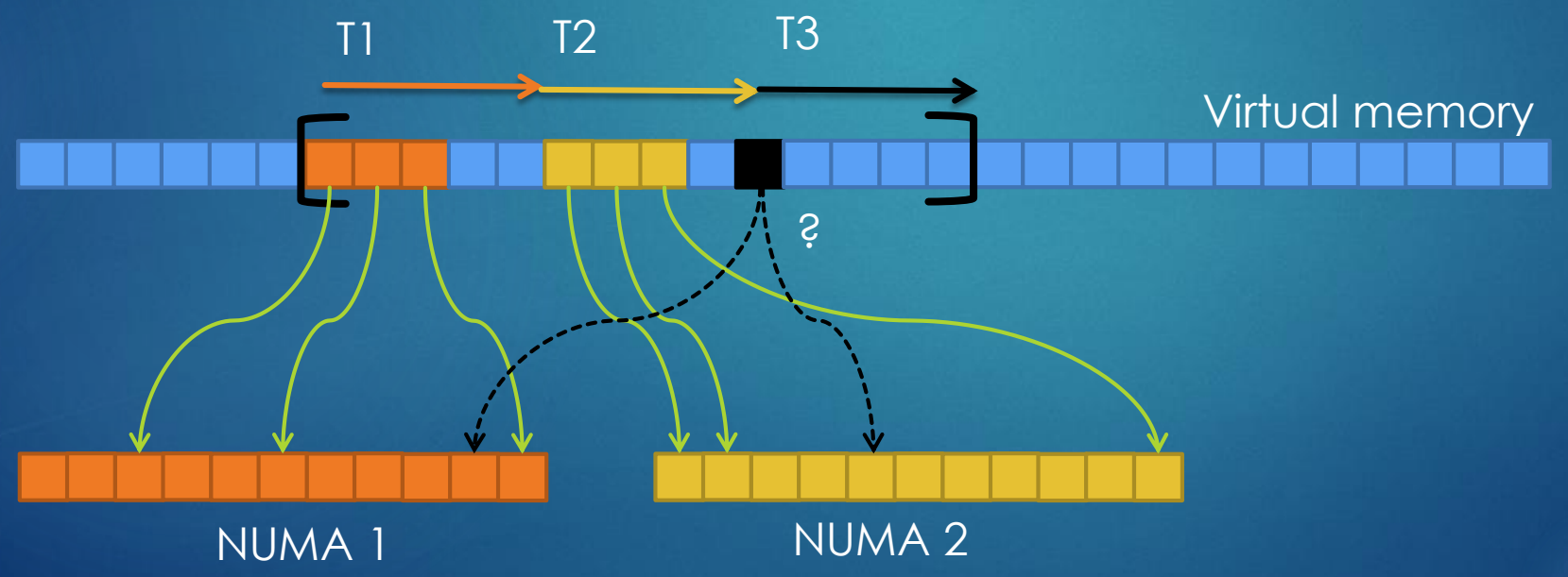
5.6 G	gomp_malloc
959.4 M	Domain::setTiles() [clone ._omp_…
206.1 M	Soa::Soa(int, int, int)
19.5 M	ThreadBuffers::ThreadBuffers(int…
1.4 M	Domain::setTiles()
773.2 K	Tile::initTile(Soa*)
769.0 K	main
128.3 K	Domain::createTestCase()
72.9 K	Matrix2<int>::listMortonIdx()

/data/svalat/Projects/Hydro/HydroC/HydroCplusMPI/ThreadBuffers.cpp | ThreadBuffers::ThreadBuffers(int, in

```
                22
                23  ThreadBuffers::ThreadBuffers(int32_t xmin, int32_t xmax, int32_t
                24  {
                25      int32_t lgx, lgy, lgmax;
                26      lgx = (xmax - xmin);
                27      lgy = (ymax - ymin);
                28      lgmax = lgx;
                29      if (lgmax < lgy)
                30          lgmax = lgy;
                31
232.2 K         32      m_q = new Soa(NB_VAR, lgx, lgy);
121.4 K         33      m_qxm = new Soa(NB_VAR, lgx, lgy);
121.4 K         34      m_qxp = new Soa(NB_VAR, lgx, lgy);
230.7 K         35      m_dq = new Soa(NB_VAR, lgx, lgy);
121.4 K         36      m_qleft = new Soa(NB_VAR, lgx, lgy);
121.4 K         37      m_qright = new Soa(NB_VAR, lgx, lgy);
121.4 K         38      m_qgdnv = new Soa(NB_VAR, lgx, lgy);
                39
11.5 M          40      m_c = new Matrix2 < real_t > (lgx, lgy);
7.0 M           41      m_e = new Matrix2 < real_t > (lgx, lgy);
                42
```

Implicit binding : first touch

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

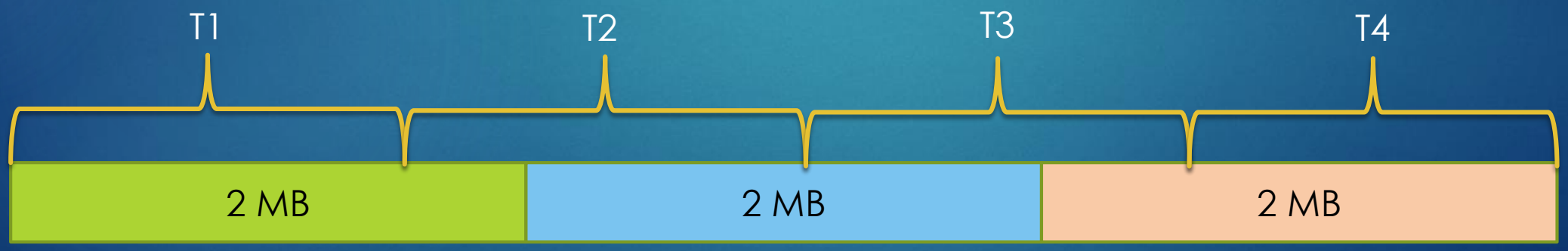


OMP and huge pages

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

```
#pragma omp parallel for
for (int i = 0 ; i < SIZE ; i++)
        array[i] = 0;

#pragma omp parallel for
for (int i = 0 ; i < SIZE ; i++)
        array[i]++;
```



GUI and example

NUMPROF

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

Reminder on NUMA

Details per thread

Numaprof | Home | Threads | Details | Sources | Assembler | Help

← « 1 2 3 4 5 6 7 8 9 10 » →

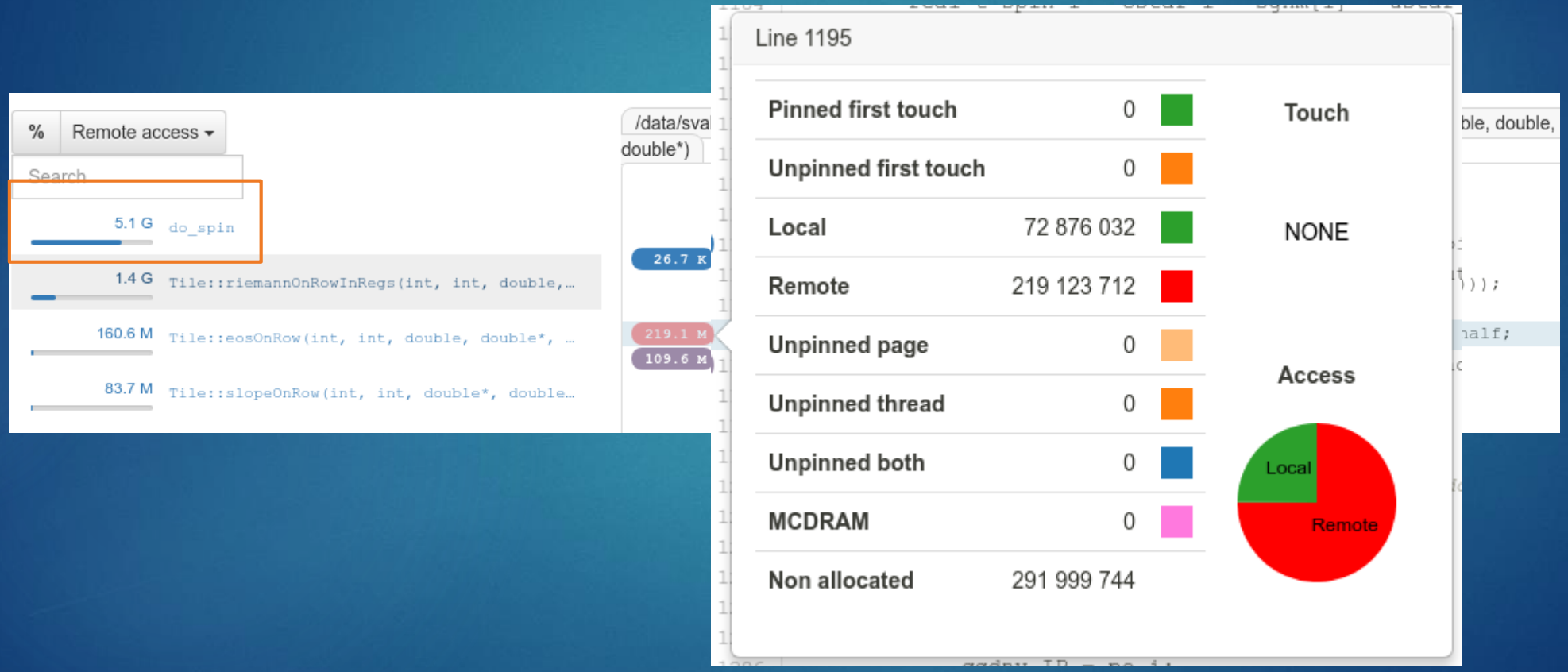
5

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	Pinned 1,024.00 — 100%
Accesses	33% — 53% — 13%
Accsses	0 1 2 3 — Unpinned
Pinning log	At 64.31%, pin thread on node 0 At 64.31%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND

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	100%
Accesses	53% — 47%
Accsses	0 1 2 3 — Unpinned
Pinning log	At 64.32%, pin thread on node 1 At 64.32%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND

Remaining consts

Sébastien Valat - COLOC - 2018
27/08/2018



Parallel allocations

Sébastien Valat - COLOC - 2018
27/08/2018

- 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);
}
```

Typical OpenMP mistake

- 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 !

Want to link to allocation site

- 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)

Similar to kernel page table

- 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**

Extracted metrics

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

What is NUMA ?

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