MALT
A Malloc Tracker

NUMAPROF
A NUMA profiler

Allocations mémoires,
Pourquoi et comment profiler ?

gray-scott reloaded / LAPP - Annecy / 07-2024

Sébastien Valat

Plan

1. The **memory problem**
2. Exiting **tools**
3. **MALT**
4. **NUMA** : Non Uniform Memory Access
5. Shortly **NUMAPROF**

Vi-HPS

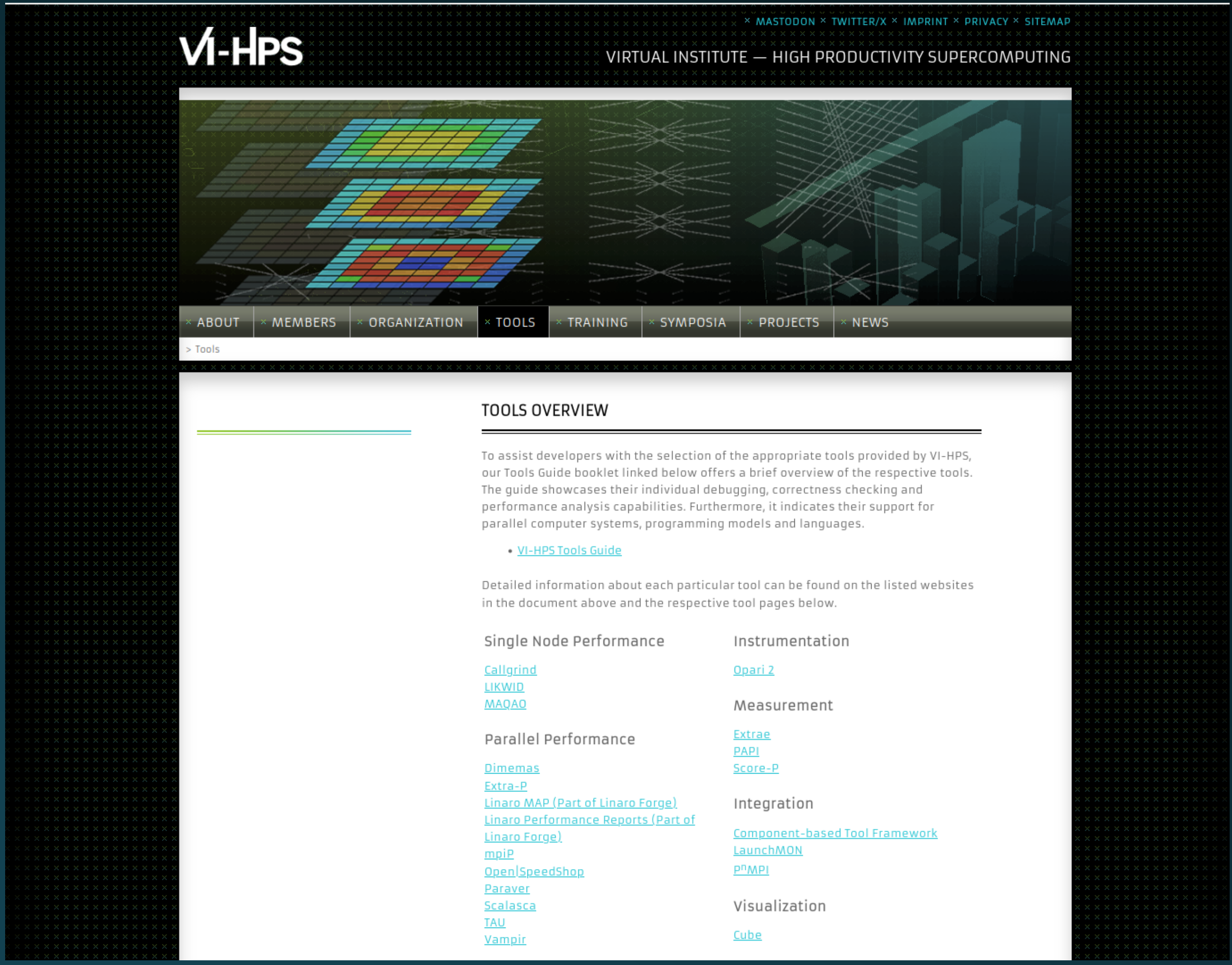
Virtual Institute — High Productivity Supercomputing

Vi-HPS

https://www.vi-hps.org/

SCAN ME

Next : **4 – 6 september 2024** / **Ostrava**



Origin of the tools

- **PhD**. On **memory management** for **HPC** (at CEA / UVSQ)
- **MALT** : post-doc at Versailles :

Exascale computing research

- **NUMAPROF** : side project post-doc work at :

CERN

- **URL :**

Get both on :
https://memtt.github.io/

Motivations

- Lot of **issues** today :
  - **Huge** memory **space** to **manage** (~TB of memory)
  - **Lot more** distinct **allocations** (e.g. 75M in 5 minutes)
  - **Multi-threaded** : 256 threads
  - **Hidden** into large (**huge**) C/C++/Fortran **codes** (**~1M** lines)

- Access:
  - **NUMA** (Non Uniform Memory Access)
  - **Memory wall !**

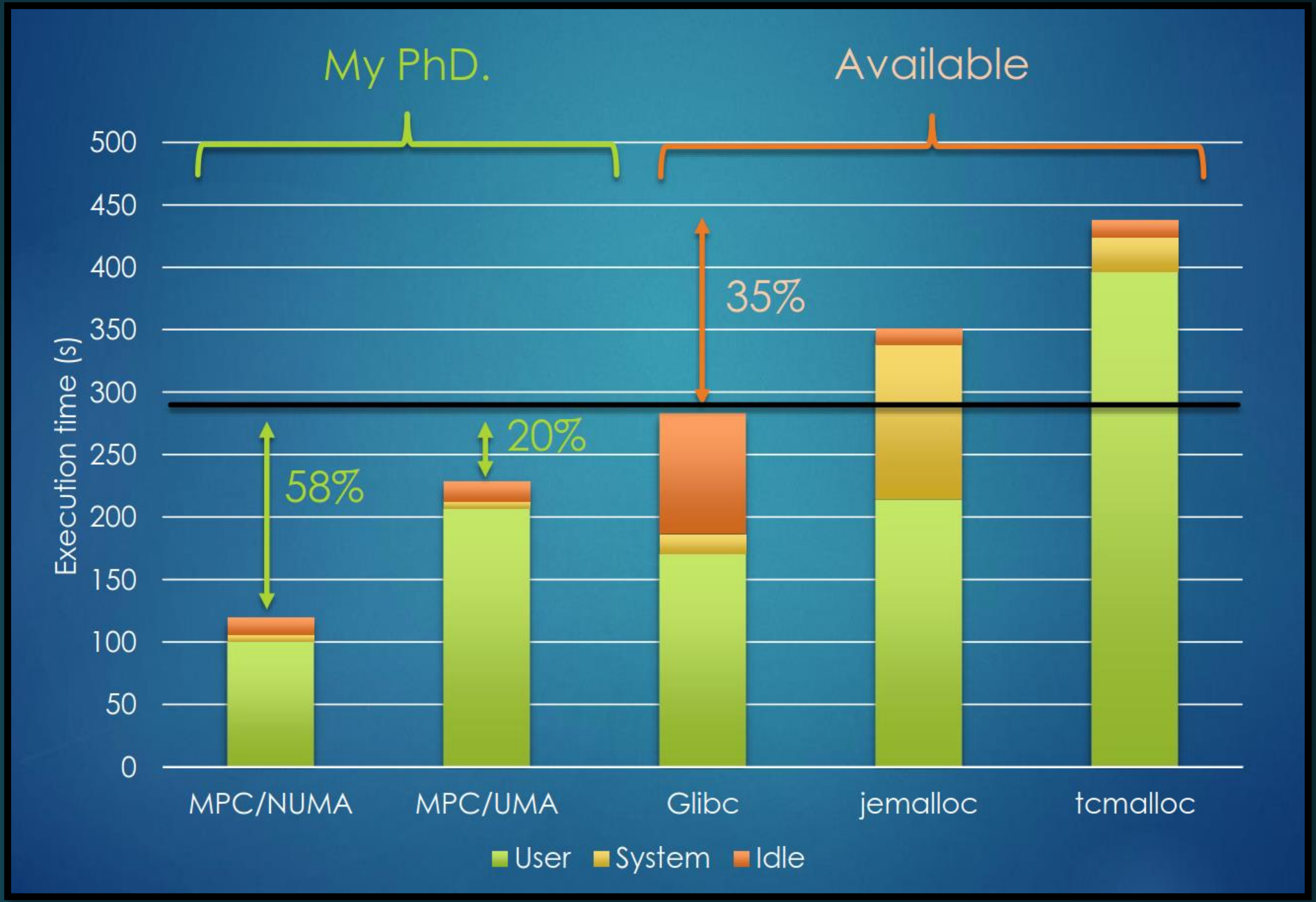
SuperMicro, 2024, 5019P-MT
1 TB ~ 10 000€

Sébastien Valat - INRIA / LJK / UGA

Key today

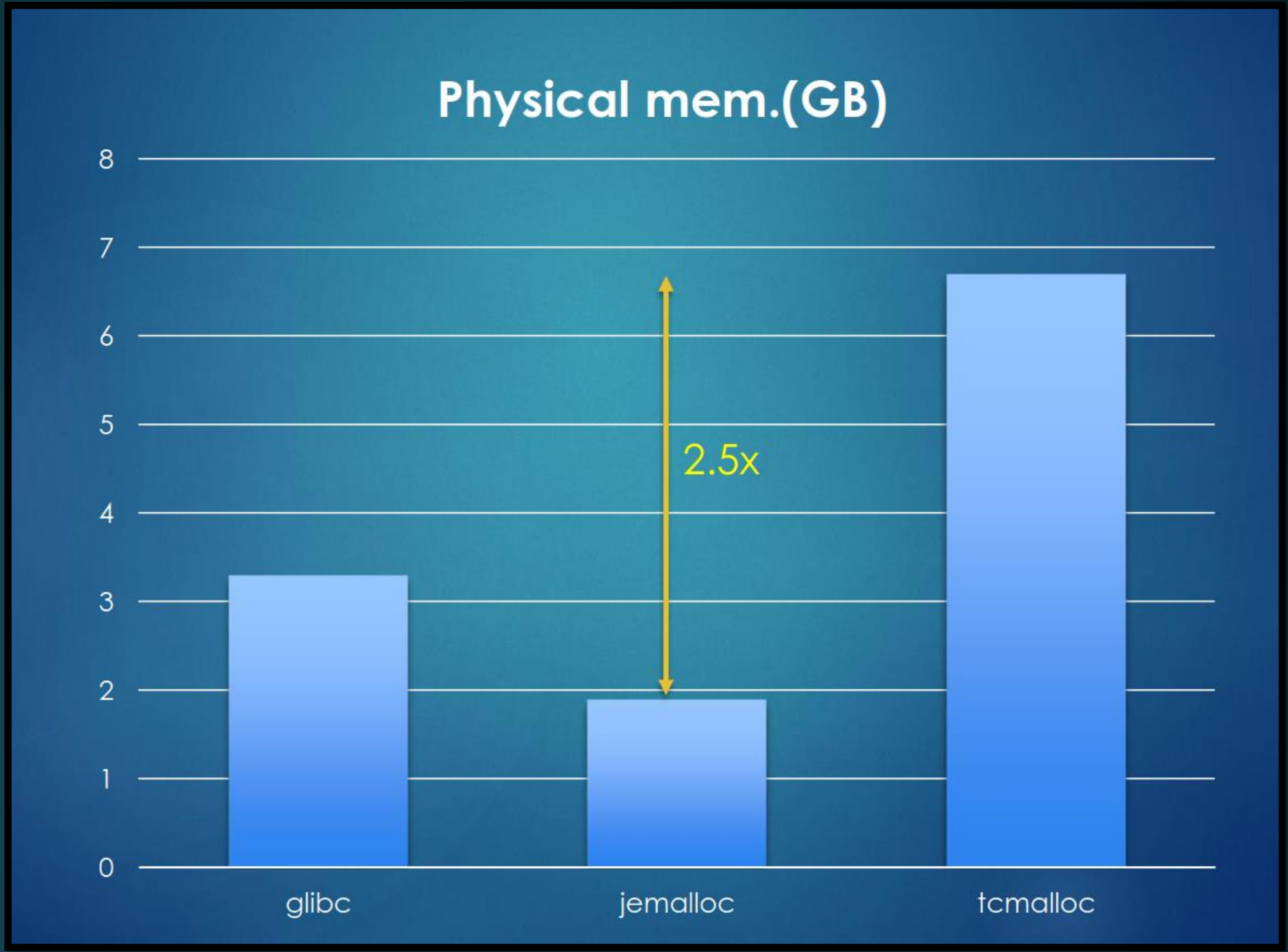
You need to
well understand
the memory behavior
of your (HPC)
application !

Eg: **>1M lines** C++ simulation. On **128** cores / **16** NUMA CPUs



Sébastien Jean Valat. Contribution à l'amélioration des méthodes d'optimisation de la gestion de la mémoire dans le cadre du Calcul Haute Performance. (tel-01253537)

Same about **memory consumption** on 12 cores



Sébastien Jean Valat. Contribution à l'amélioration des méthodes d'optimisation de la gestion de la mémoire dans le cadre du Calcul Haute Performance. (tel-01253537)

MALT

Malloc Tracker

Goal

- We have **profiling tool** for **timing** (eg. **Valgrind** or **vtune**)
- But for **memory usage** ?
- Memory can be an issue :
  - Failed to run (or swap) due to **lack of memory resource**.
  - **Performance impact** of memory management functions.
  - Impact due to **memory layout**.

Some issue examples

We want to help searching :

- **Where** memory is allocated.
- **Properties** of allocated chunks.
- **Bad** allocation **patterns** for performance.
- **Leaks**
- **Global variables** (TLS)

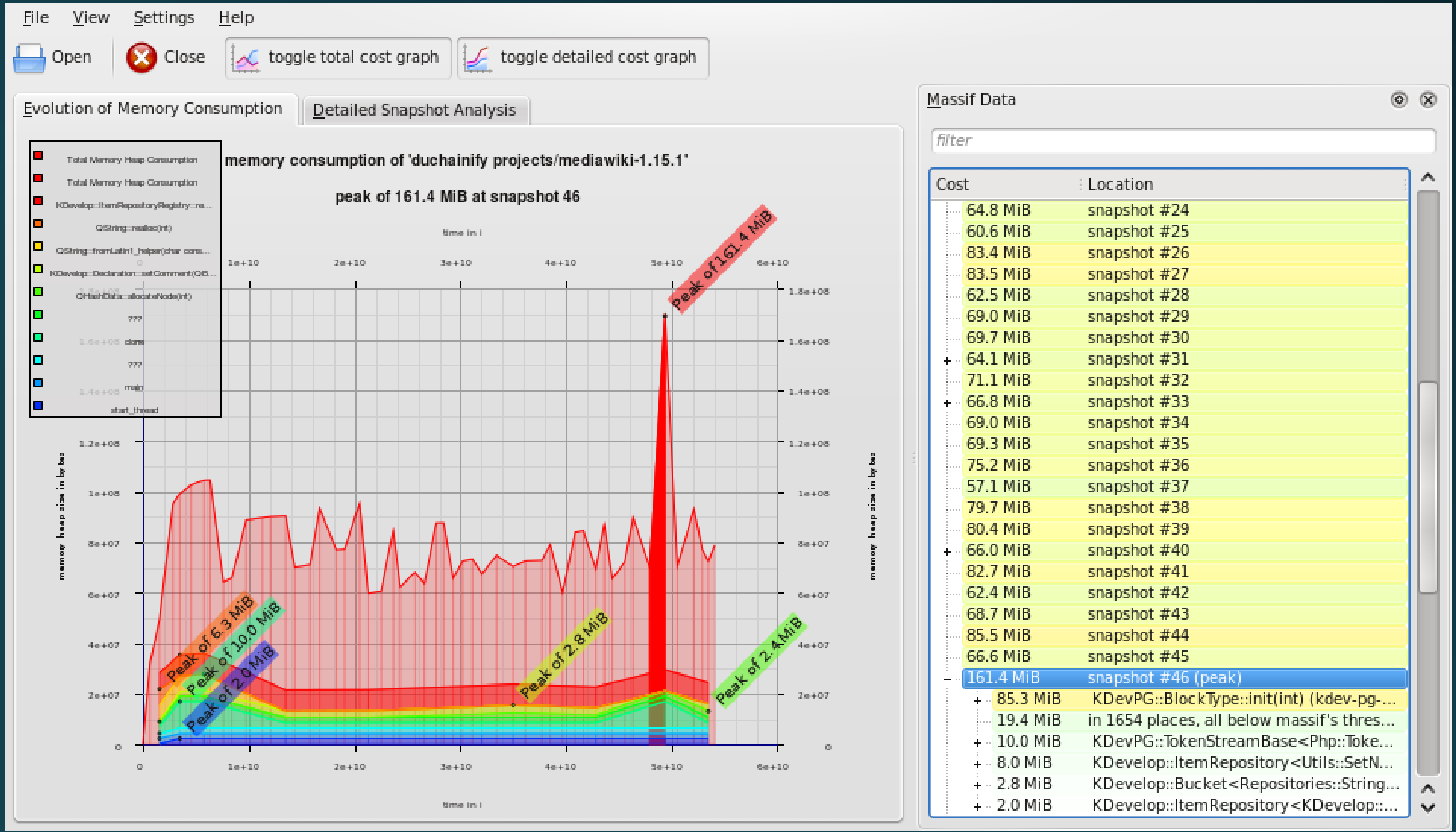
Some issue examples

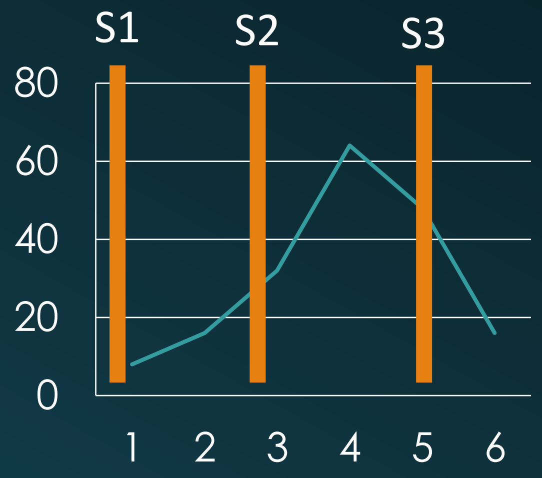
```
Int gblVar[SIZE];
int * func(int size)
{
		child_func_with_allocs();
		void * ptr = new char[size];
		double* ret = new double[size*size*size];
		for (.....)
		{
			double* buffer = new double[size];
			//short and quick do stuff
			delete [] buffer;
		}
		return ret;
}
```

- Global **variables** or **TLS**
- **Indirect** allocations
- **Leak**
- Might lead to swap for **large size**
- **Short life** allocations

Existing tools

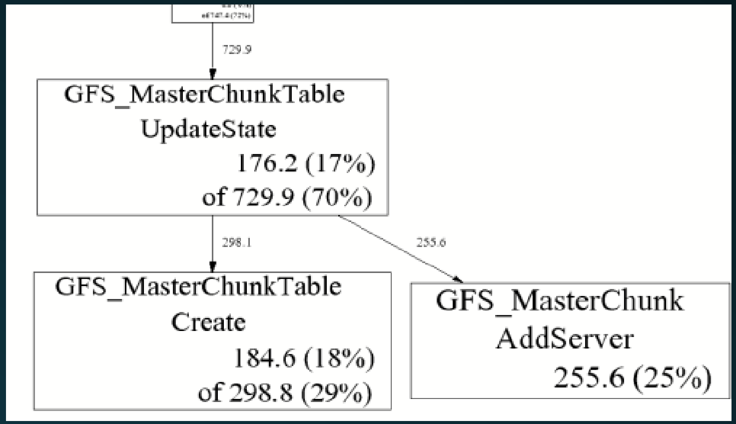
- Valgrind - **memcheck**
- Valgrind - **massif**



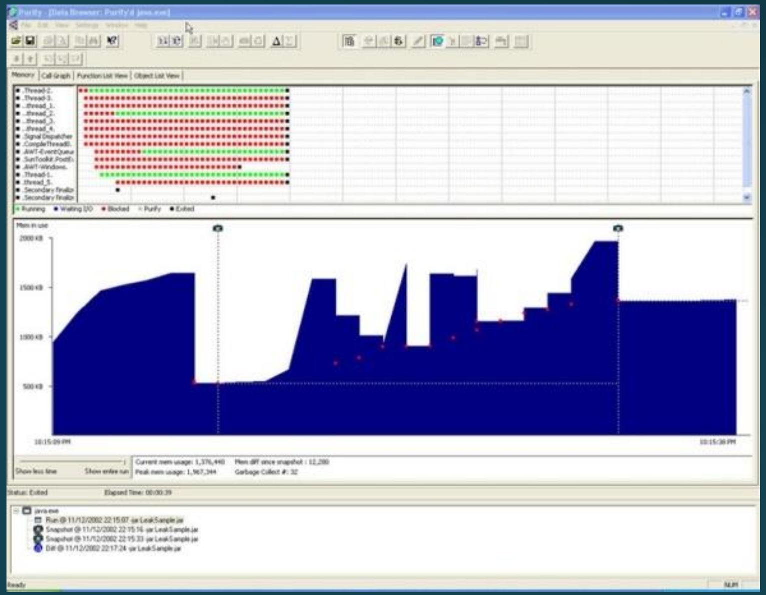


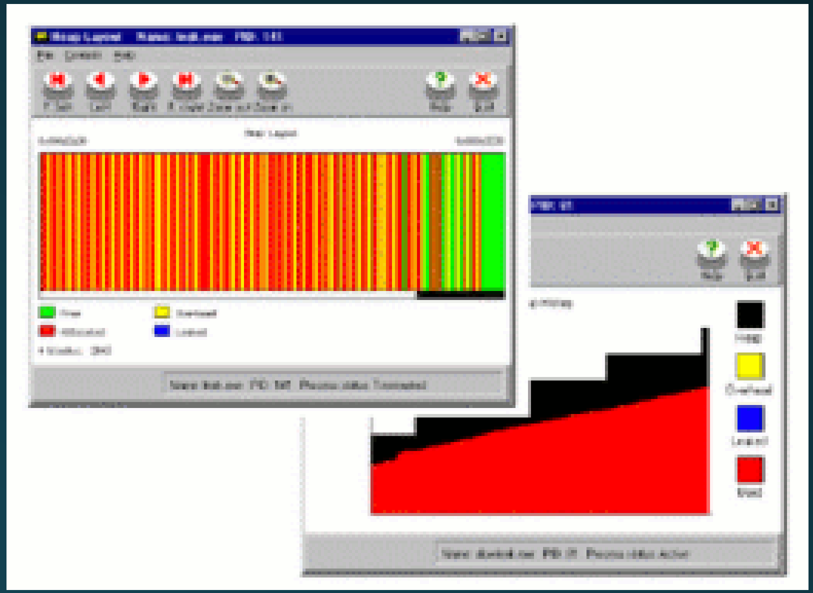
Existing tools

- **Google heap profiler** (tcmalloc):

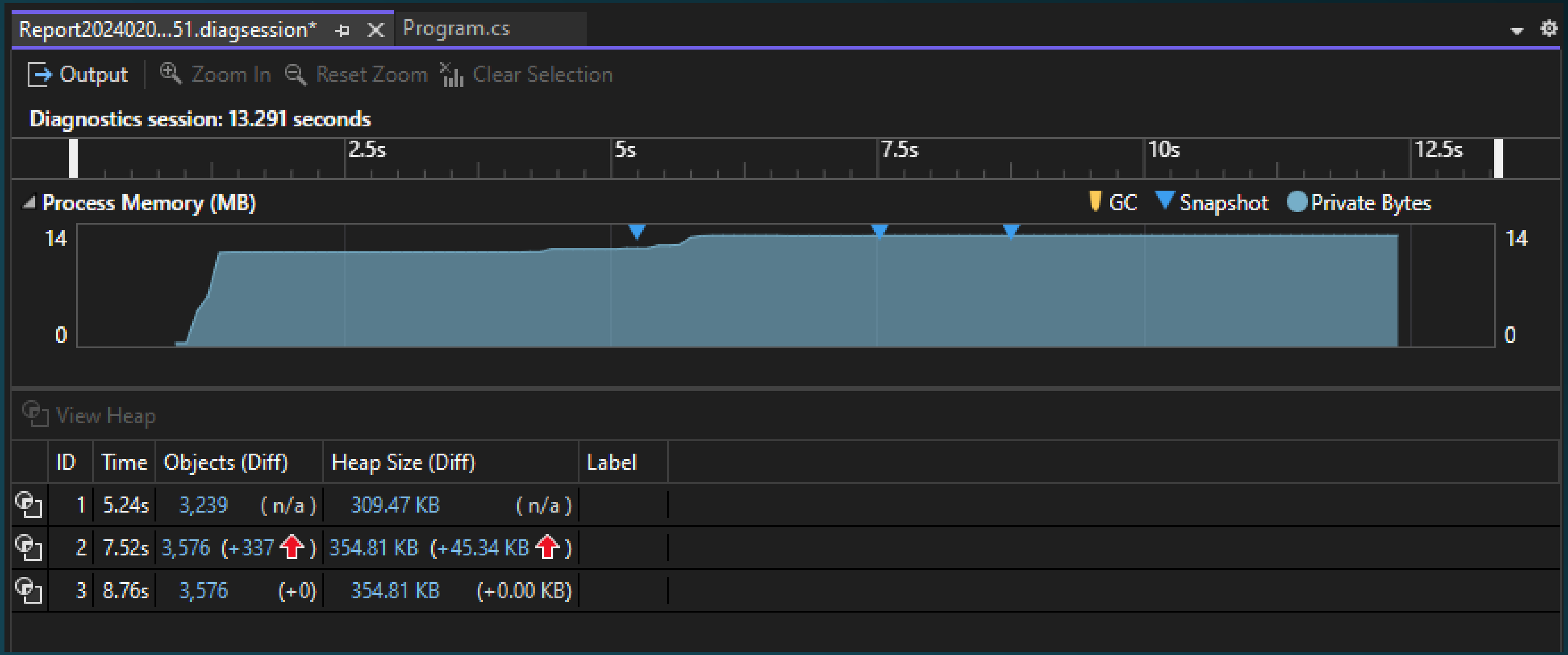


- IBM **Purify++** / Parasoft **Insure++**





Visual Studio Ultimate memory profiler



What I want to provide

- Same **approach** than **valgrind / kcachegrind**

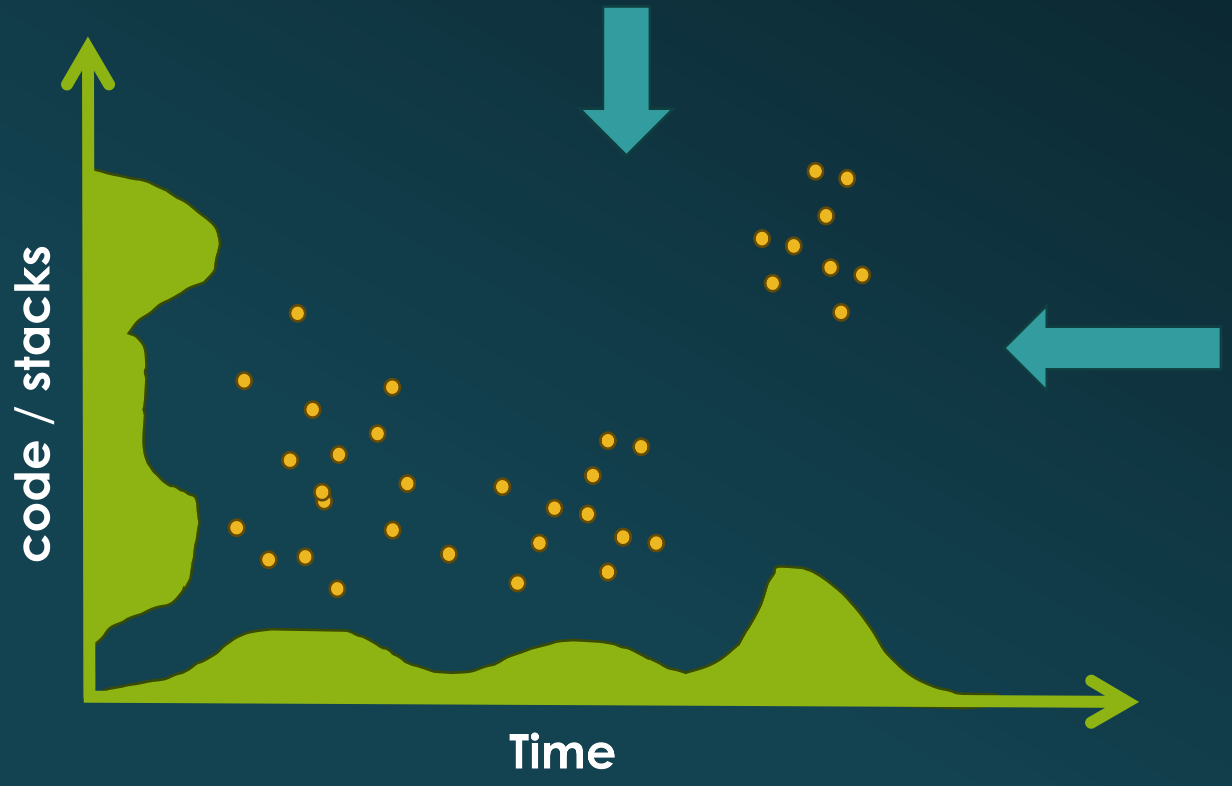
- **Mapped allocations on sources lines**

- For memory resource usage :
  - **Memory leaks** (malloc without free)
  - **Peak** and total allocated memory

- For performance :
  - Allocation **count**
  - Allocation **sizes** (min/mean/max)
  - Chunk **lifetime** (min/mean/max)

Profile based

- Project on **two axis**:
  - Over **code / stacks**
  - Over **time**



Usage

Profiling

```
malt ./mon_prog
```

→ JSON →

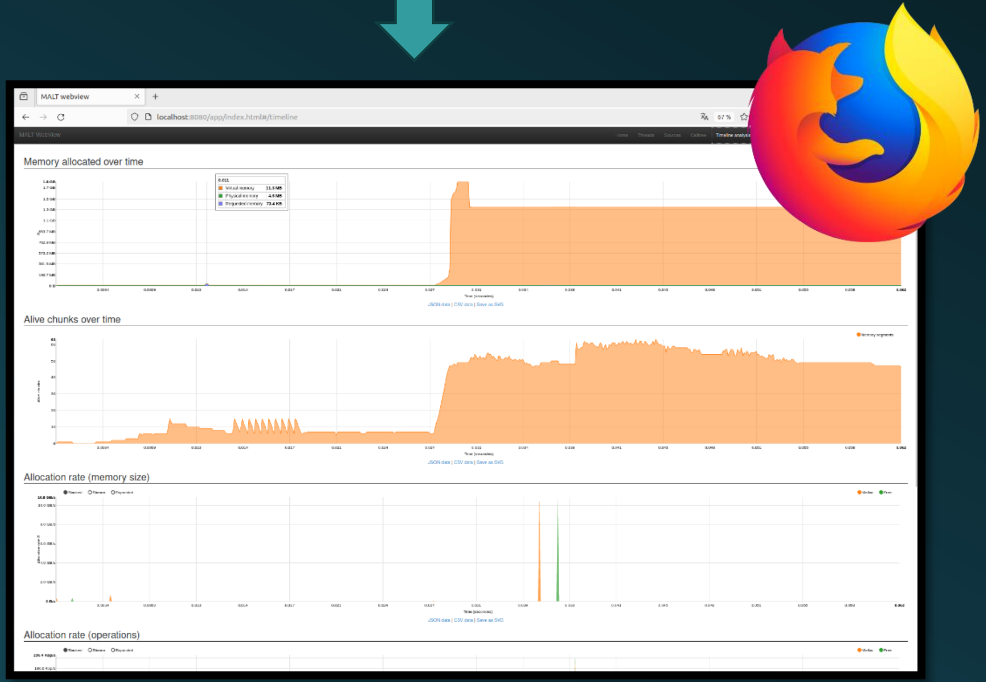
Viewing via light web server

```
malt-webview …
```

```
firefox http://localhost:8080
```

Can be forwarded via SSH

```
ssh -L8080:localhost:8080 ………
```



Profiling on a cluster

Your laptop

```
ssh -L8080:localhost:8080 ………
```

Remote cluster / server

Profiling

```
malt ./mon_prog
```

Project sources

JSON

Libraries sources

Viewing via light web server

```
malt-webview …
```

Global summary

EXECUTION TIME	PHYSICAL MEMORY PEAK	ALLOCATION COUNT	AVAILABLE PHYSICAL MEMORY
00:00:00.25	2.3 MB	379	4.1 Gb

Run description

Executable :	simple-case-finstr-linked
Commande :	`./simple-case-finstr-linked`
Tool :	matt-0.0.0
Host :	localhost
Date :	2014-11-26 22:40
Execution time :	00:00:00.25
Ticks frequency :	1.8 GHz

Global statistics

Show all details Show help

Physical memory peak	2.3 MB
Virtual memory peak	103.7 MB
Requested memory peak	[illegible]

Global summary

Global statistics

Show all details | Show help

Physical memory peak	2.3 MB
Virtual memory peak	103.7 MB
Requested memory peak	3.9 KB
Cumulated memory allocations	26.4 KB
Allocation count	379
Recycling ratio	6.7
Leaked memory	2.1 KB
Largest stack	6.0 KB
Global variables	41.2 KB
TLS variables	4.0 KB
Peak allocation rate	25.2 MB/s

Can give some hints (warnings)

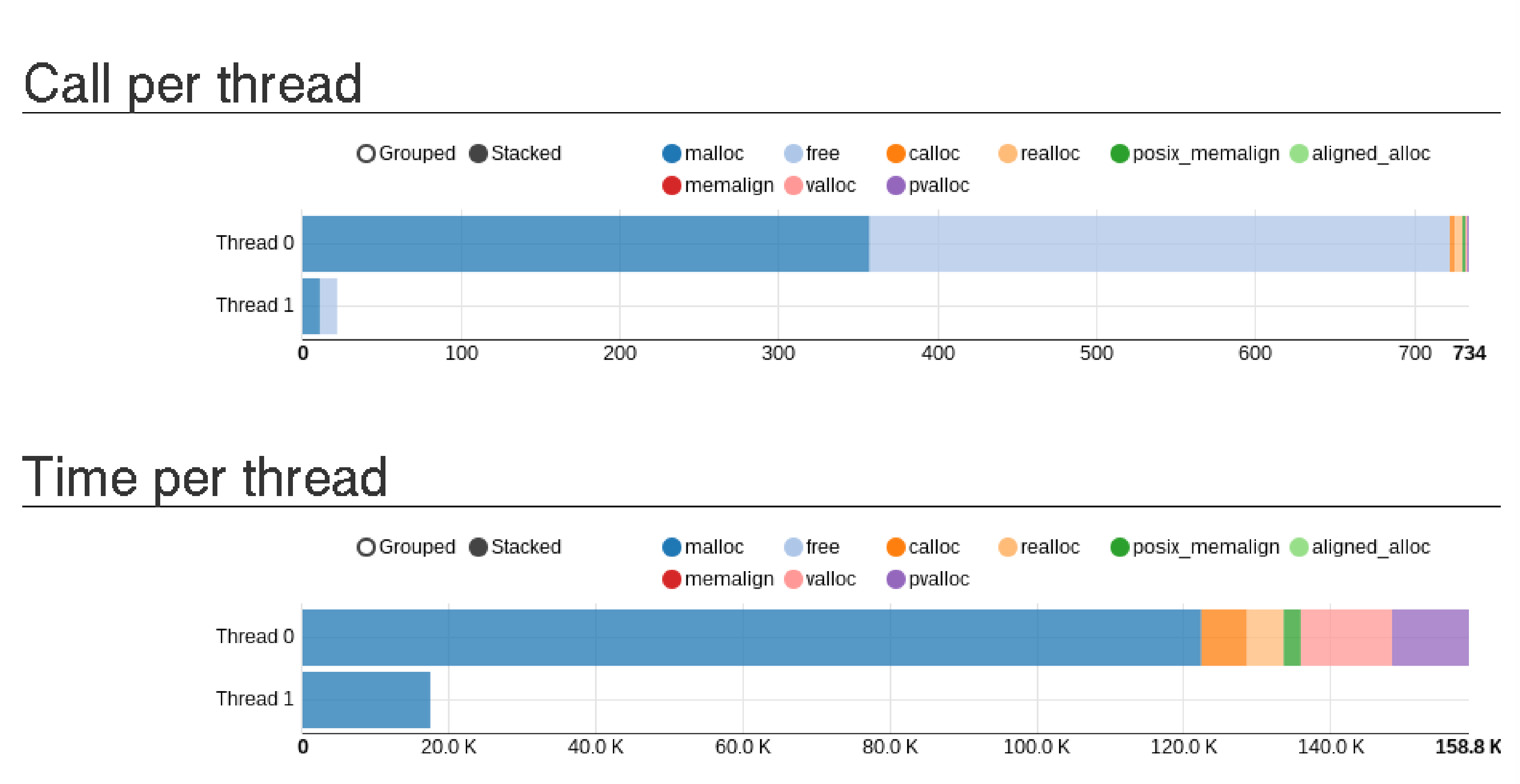
Ticks frequency :	2.9 GHz
Allocator used :	/usr/lib/x86_64-linux-gnu/libc.so.6

Global statistics

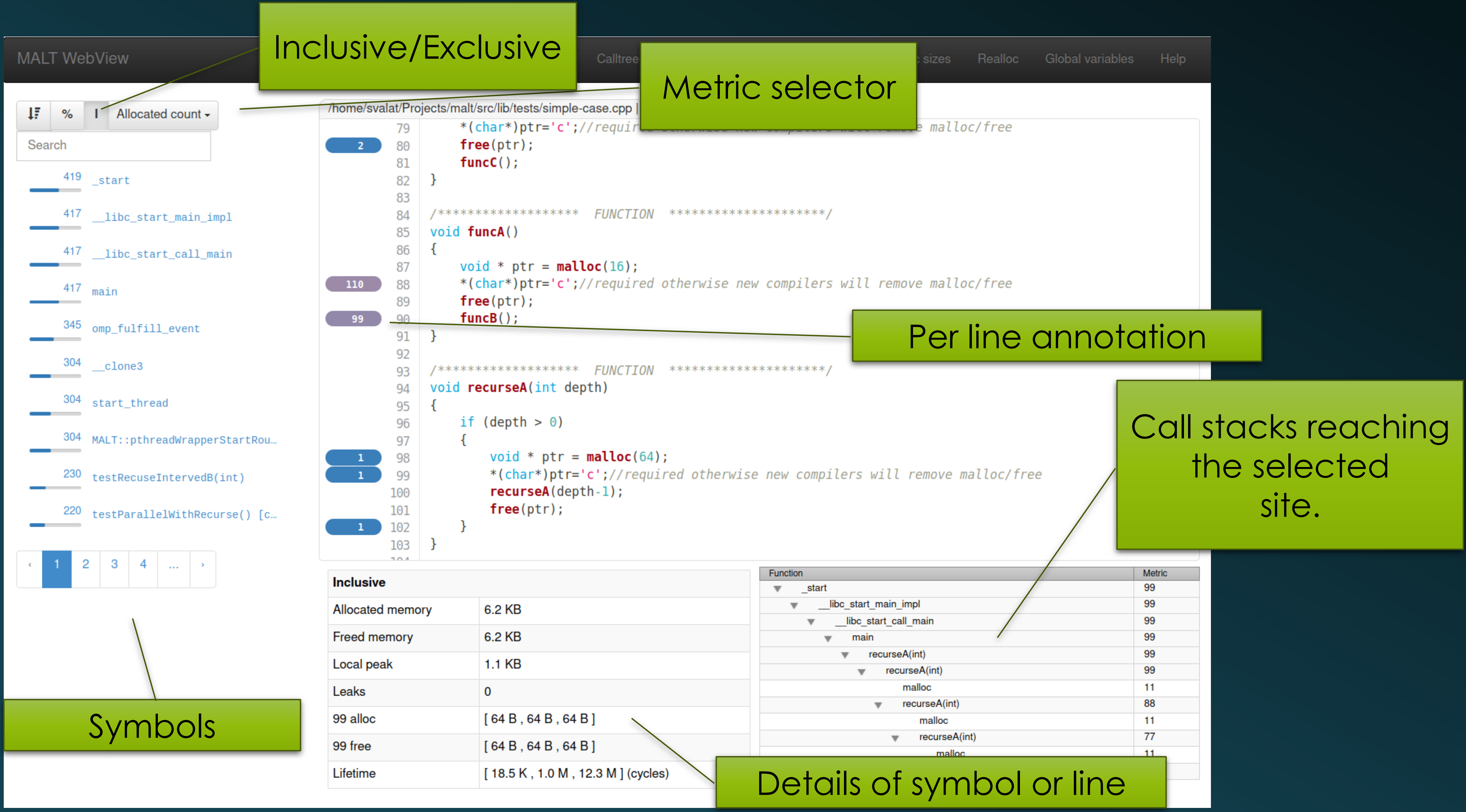
Show all details | Show help

Physical memory peak	13.8 MB
Virtual memory peak	189.4 MB
Requested memory peak	12.1 MB
Cumulated memory allocations	**144.0 GB** ⚠
Allocation count	49.2 K
Recycling ratio	**12204.8** ⚠
Leaked memory	6.1 MB
Largest stack	96 B
Global variables	43.8 KB
TLS variables	40 B
Global variable count	**789** ⚠
Peak allocation rate	309.4 GB/s

Per thread statistics

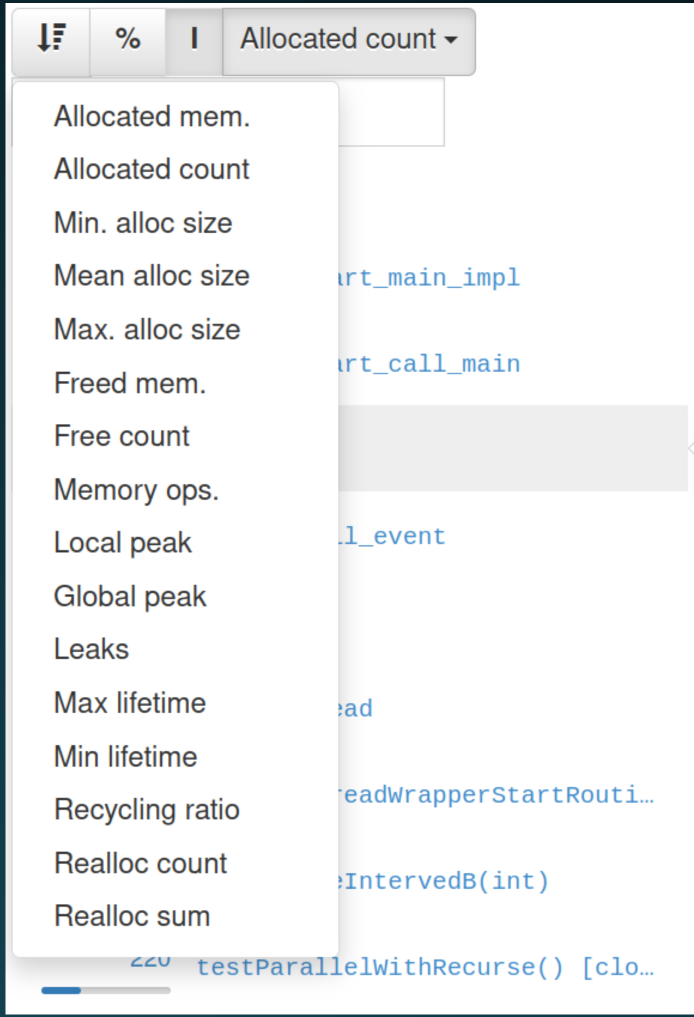


Source annotation



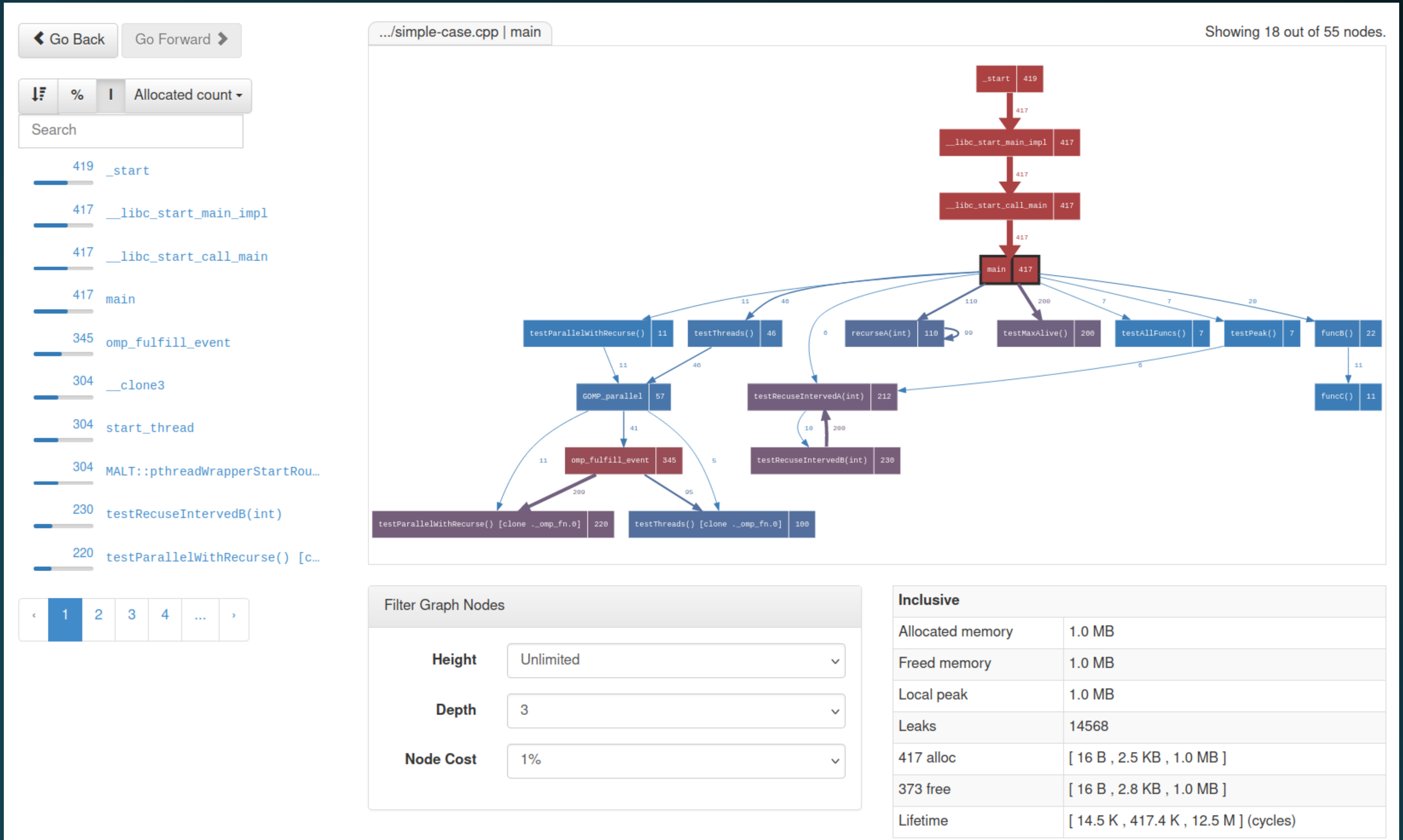
Metrics

- **Bloc sizes**
- **Frees**
- **Peaks**
- **Short life** allocations
- **Recycling**
- **Realloc**



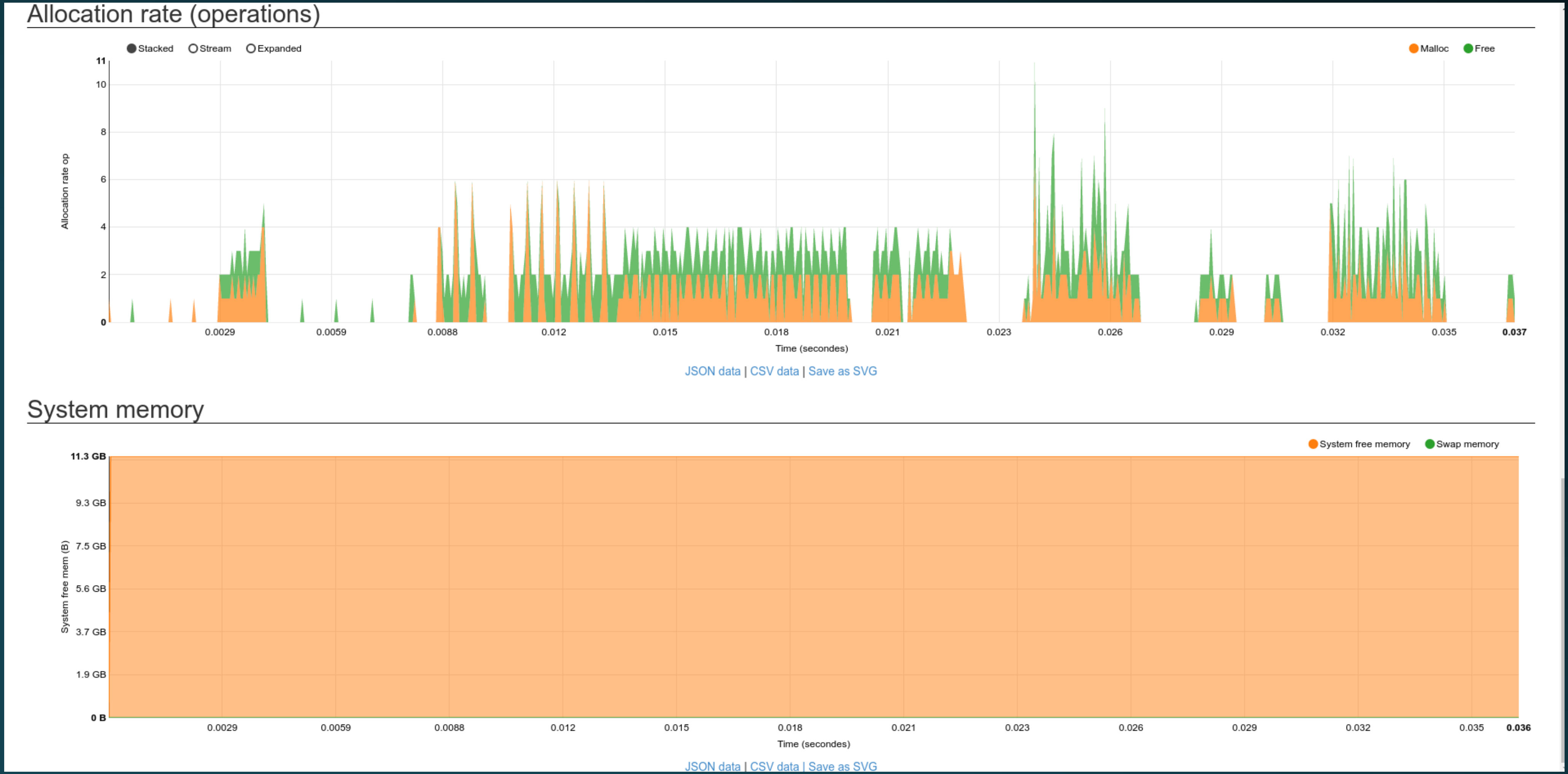
Sébastien Valat - INRIA / LJK / UGA

Call stack tree



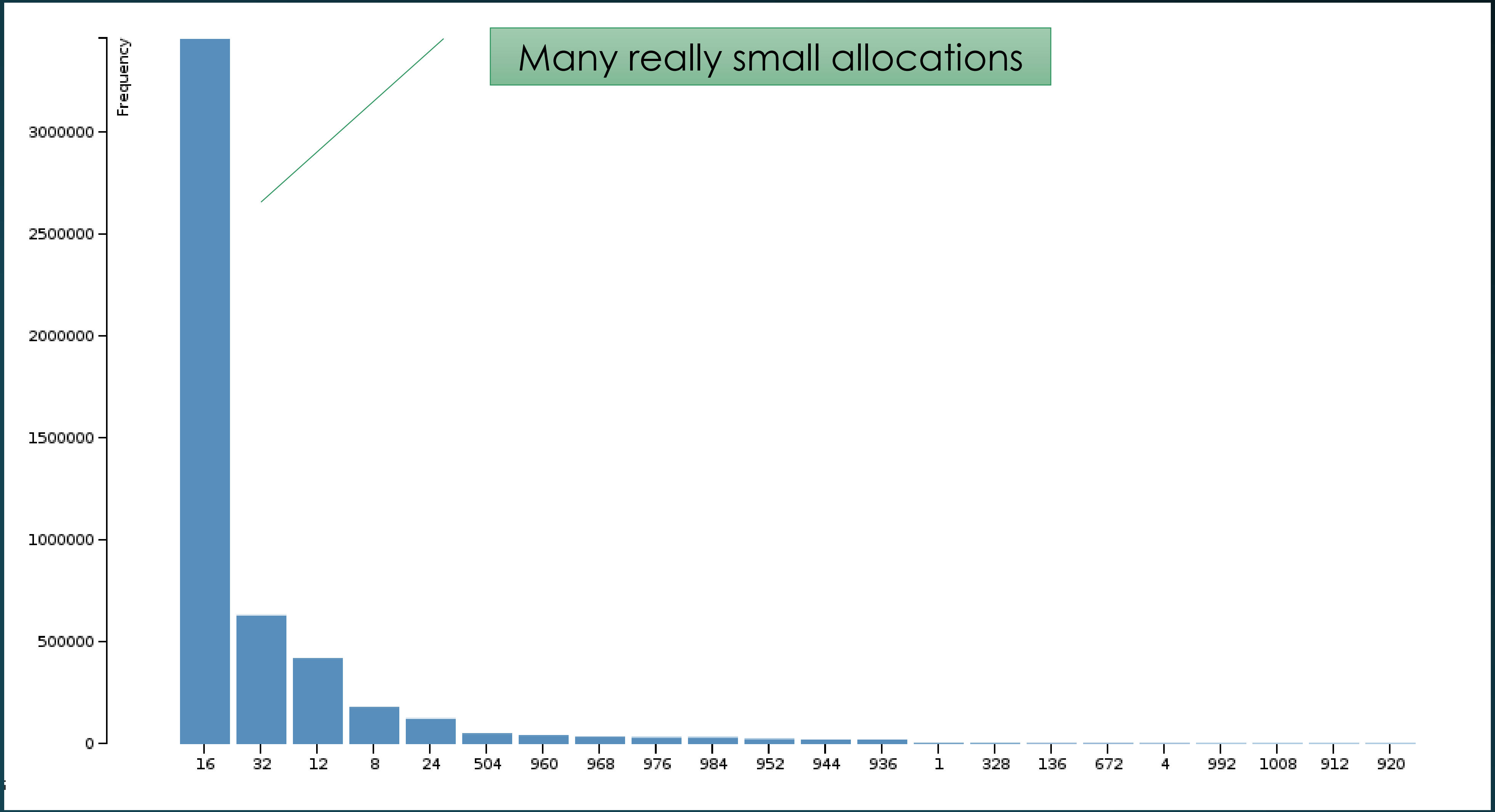
Jaffery, Syed Mehdi Raza / CERN

Time charts

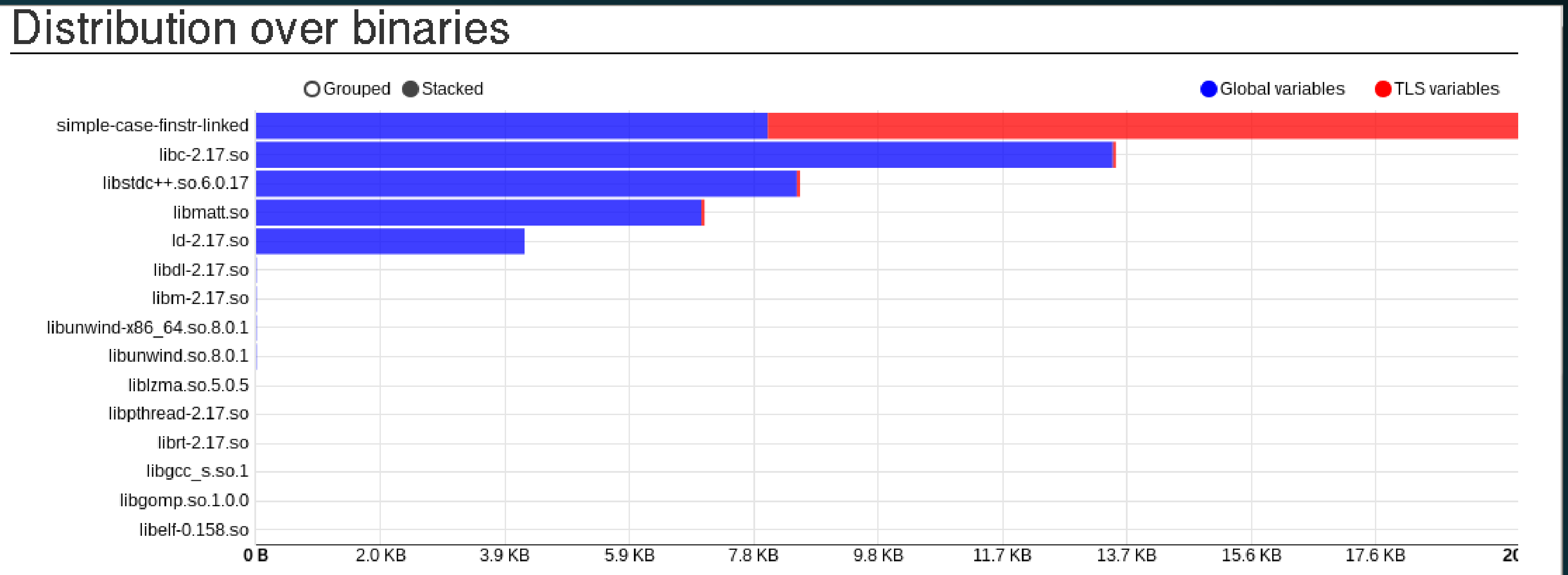

Allocation rate (operations)
Stacked
Stream
Expanded
Malloc
Free
Allocation rate op
Time (secondes)
JSON data | CSV data | Save as SVG
System memory
System free memory
Swap memory
System free mem (B)
11.3 GB
9.3 GB
7.5 GB
5.6 GB
3.7 GB
1.9 GB
0 B
Time (secondes)
JSON data | CSV data | Save as SVG

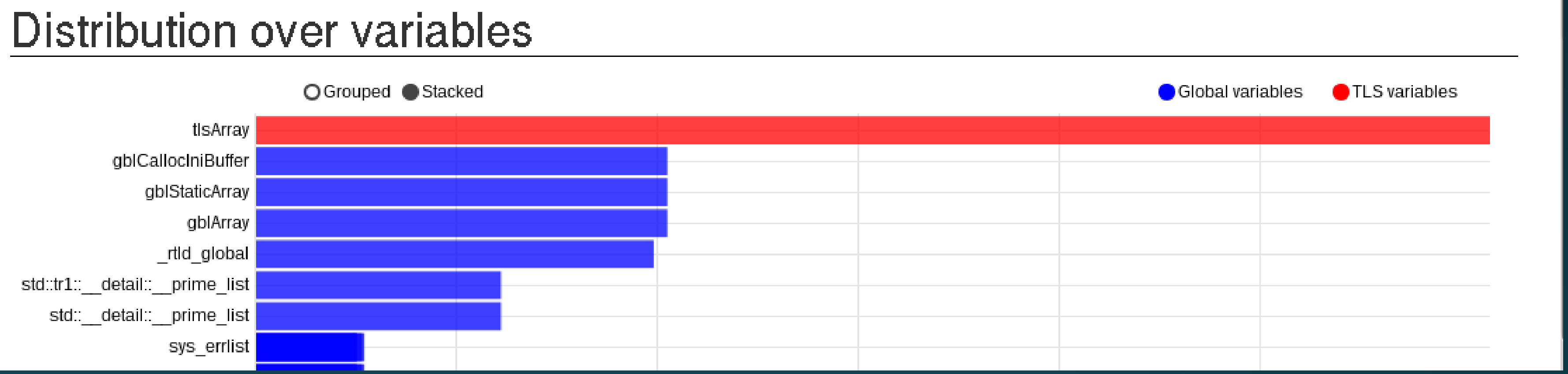
Chunk size distribution

Example from YALES2 with gfortran issue



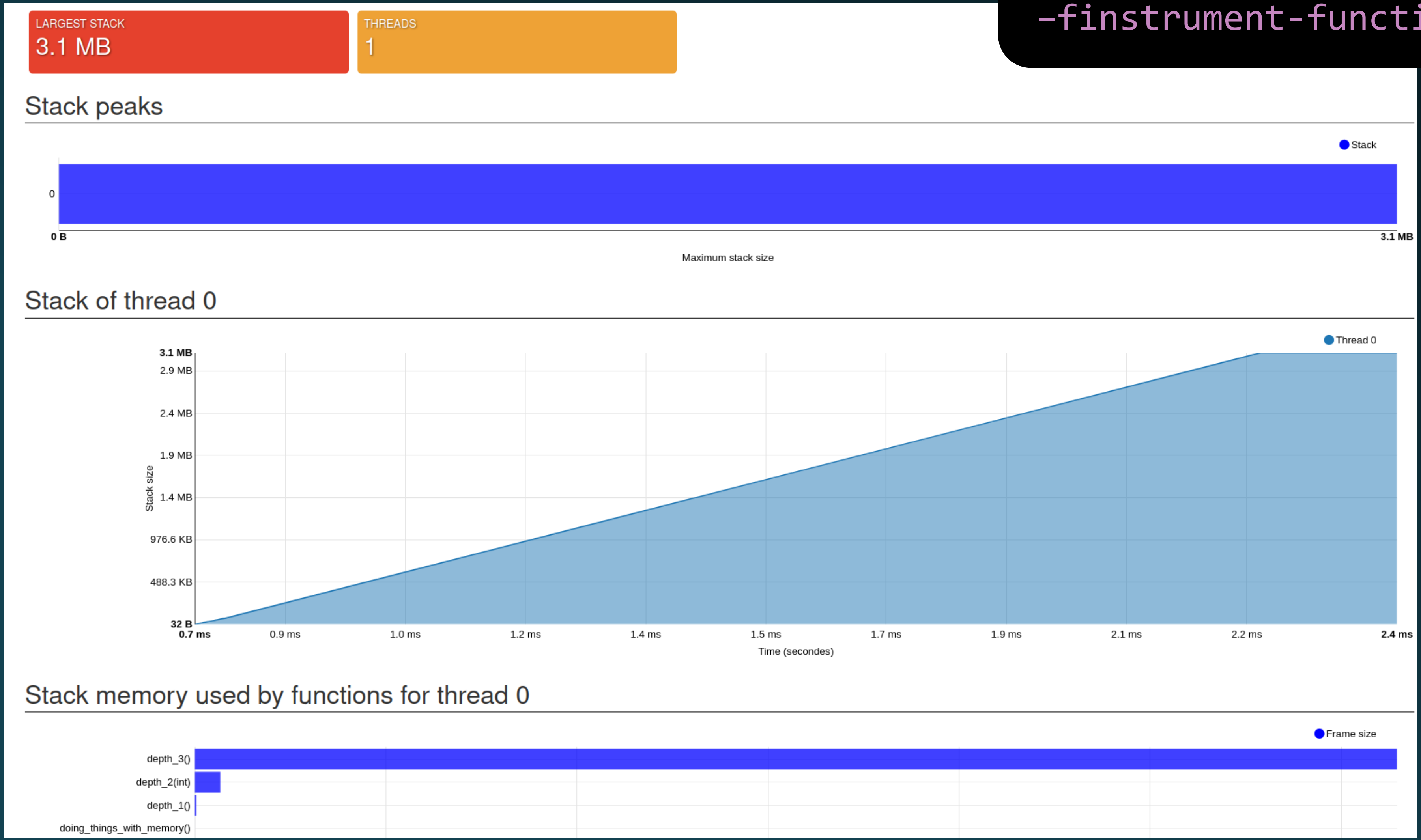
Global variables





Memory used by stacks

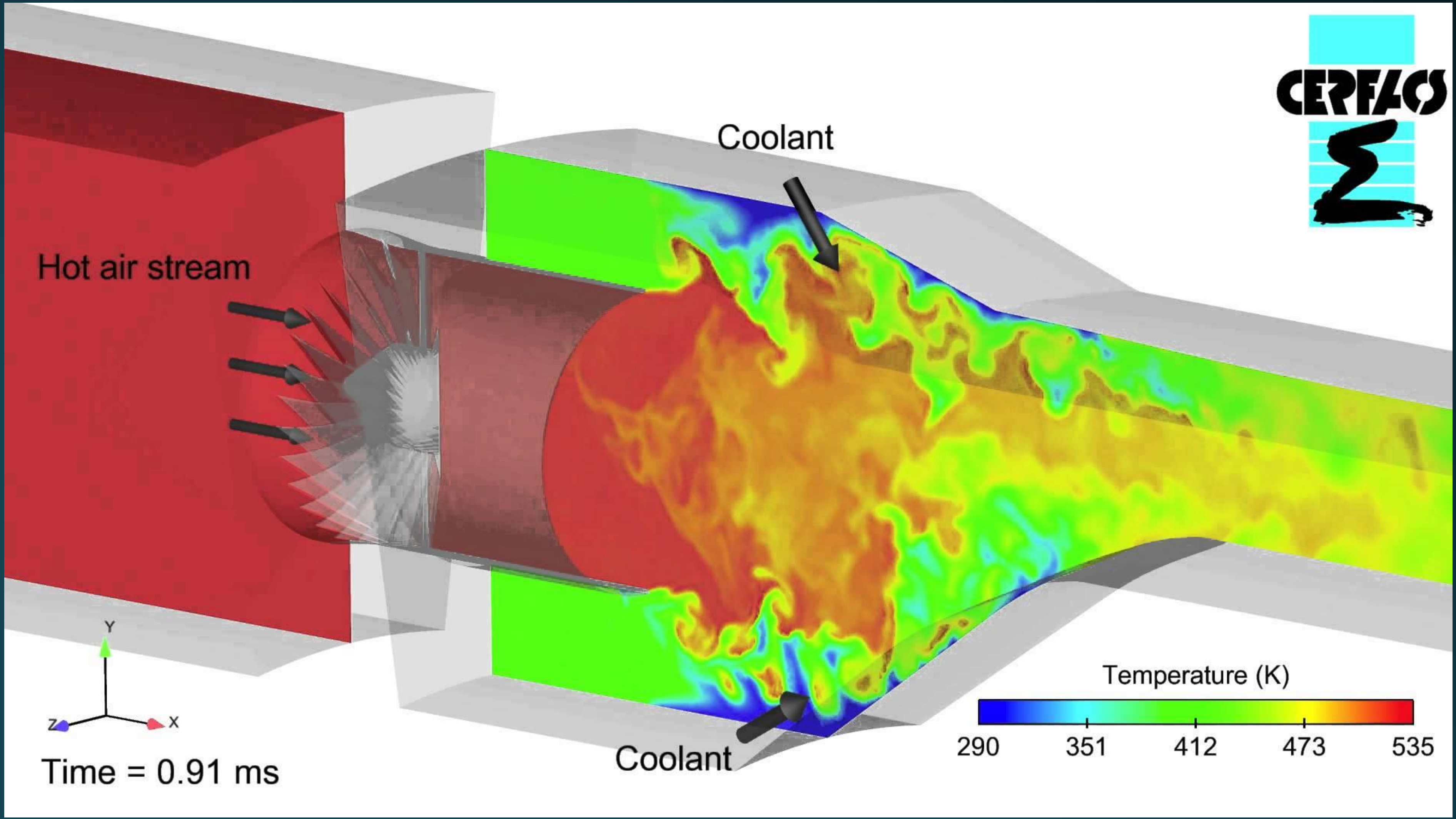
Need to recompile the app with `-finstrument-functions`



3/07/2024
Sébastien Valat - INRIA / LJK / UGA

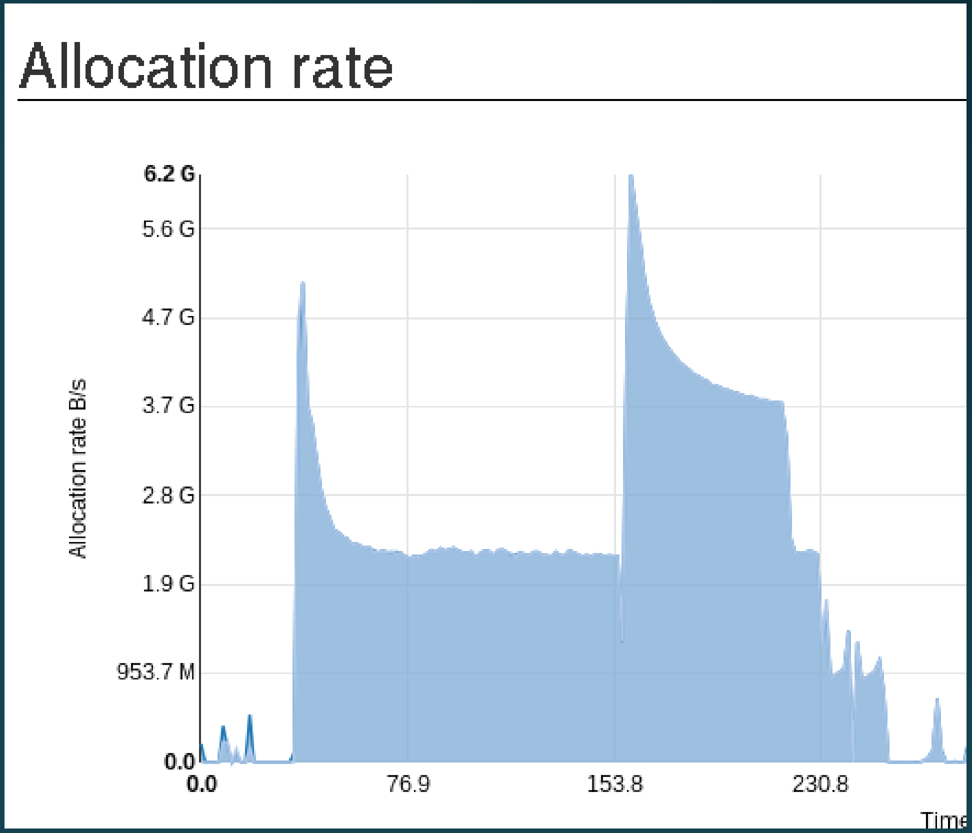
Real cases

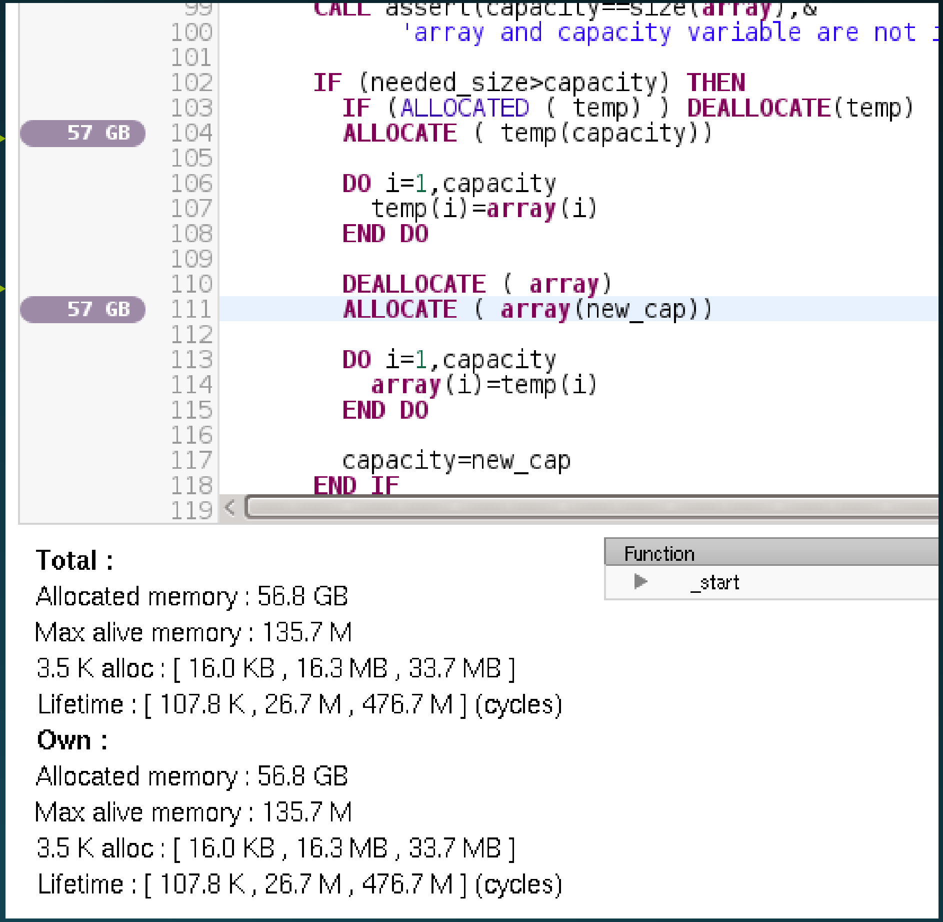
Cerfacs - AVBP- CFD simulator



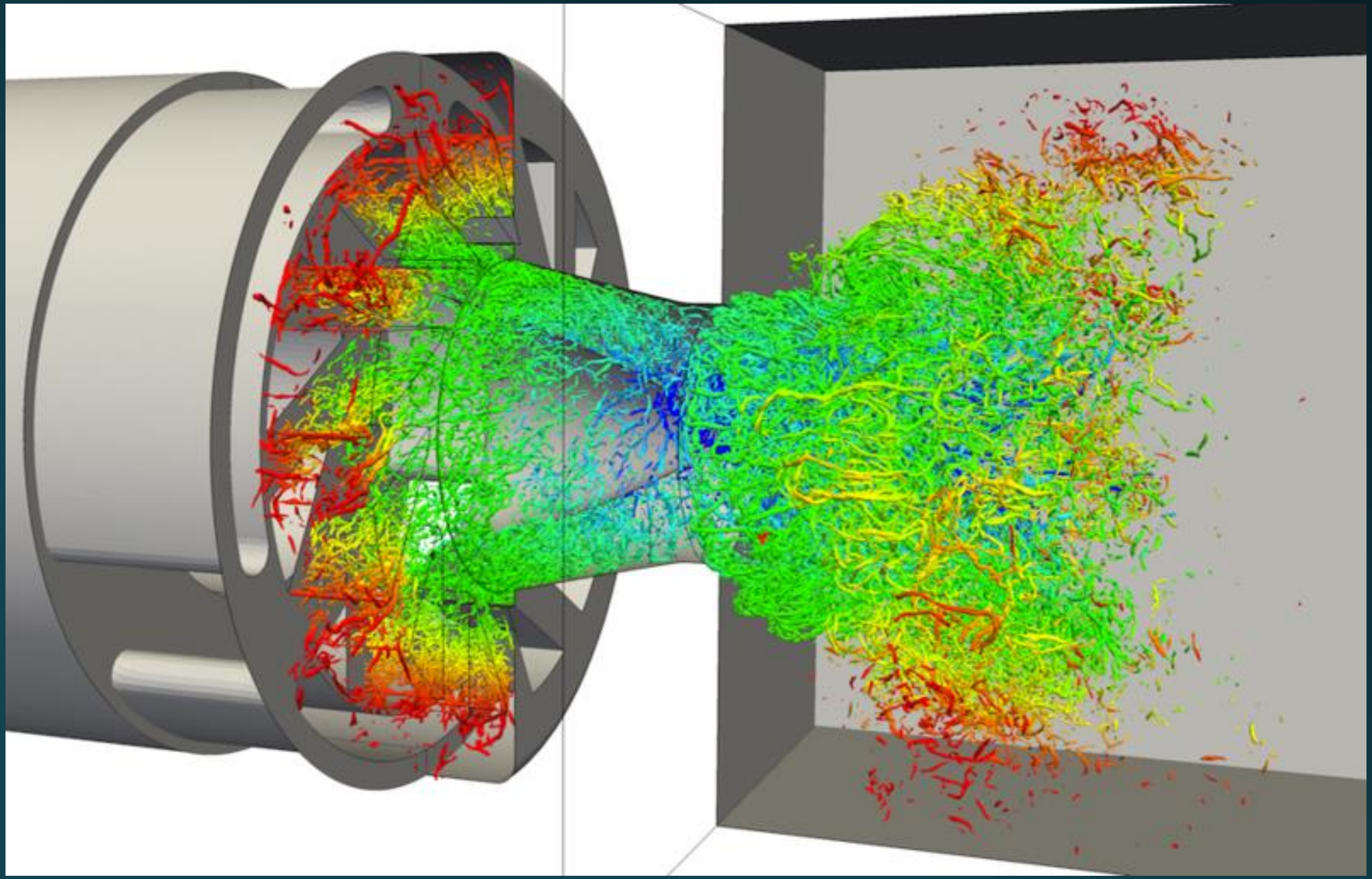
Example on AVBP init phase

- Issue with reallocation on init



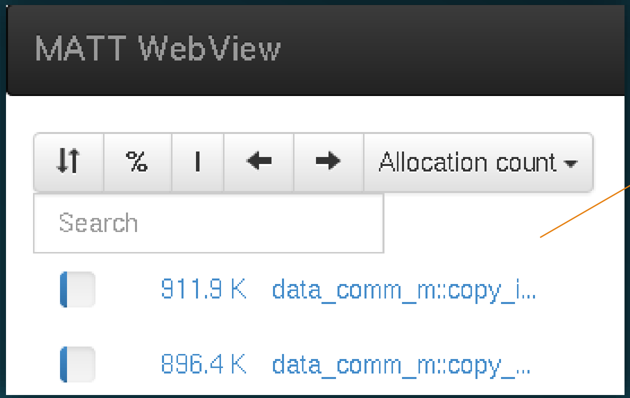


Coria – YALES2 - Combustion



Allocatable arrays on YALES2

Issue only occur with **gfortran**, **ifort** uses stack arrays.



Search intensive alloc functions

Huge number of allocation for a line programmer think it doesn't do any !

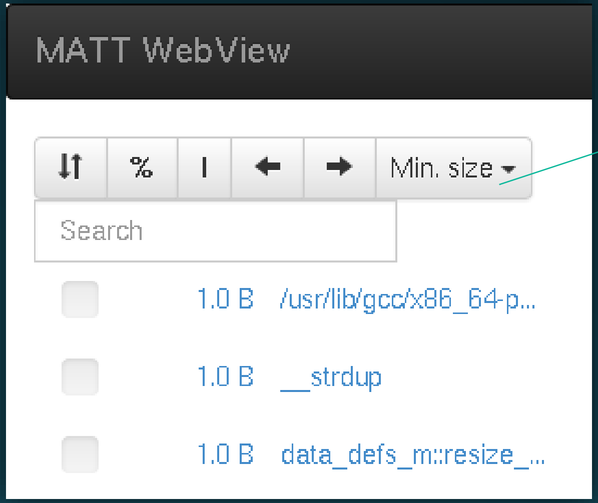
608 K

```fortran
do i=1,nitem_el_grp
   el_grp_ind = el_grp_index2int_comm_index%val(1,i)
   int_comm_ind = el_grp_index2int_comm_index%val(2,i)
   el_grp_r2%val(1:dim1,el_grp_ind) = int_comm_r2%val(1:dim1,int_comm_ind)
end do
```

Total :
Allocated memory : 9.5 MB
Freed memory : 9.5 MB
Max alive memory : 432
608.0 K alloc : [16 B , 16 B , 16 B]
608.0 K free : [16 B , 16 B , 16 B]
Lifetime : [24.5 K , 39.9 K , 37.8 M] (cycles)
Own :
Allocated memory : 9.5 MB

And mostly really small allocations !

We can found allocs of 1B !

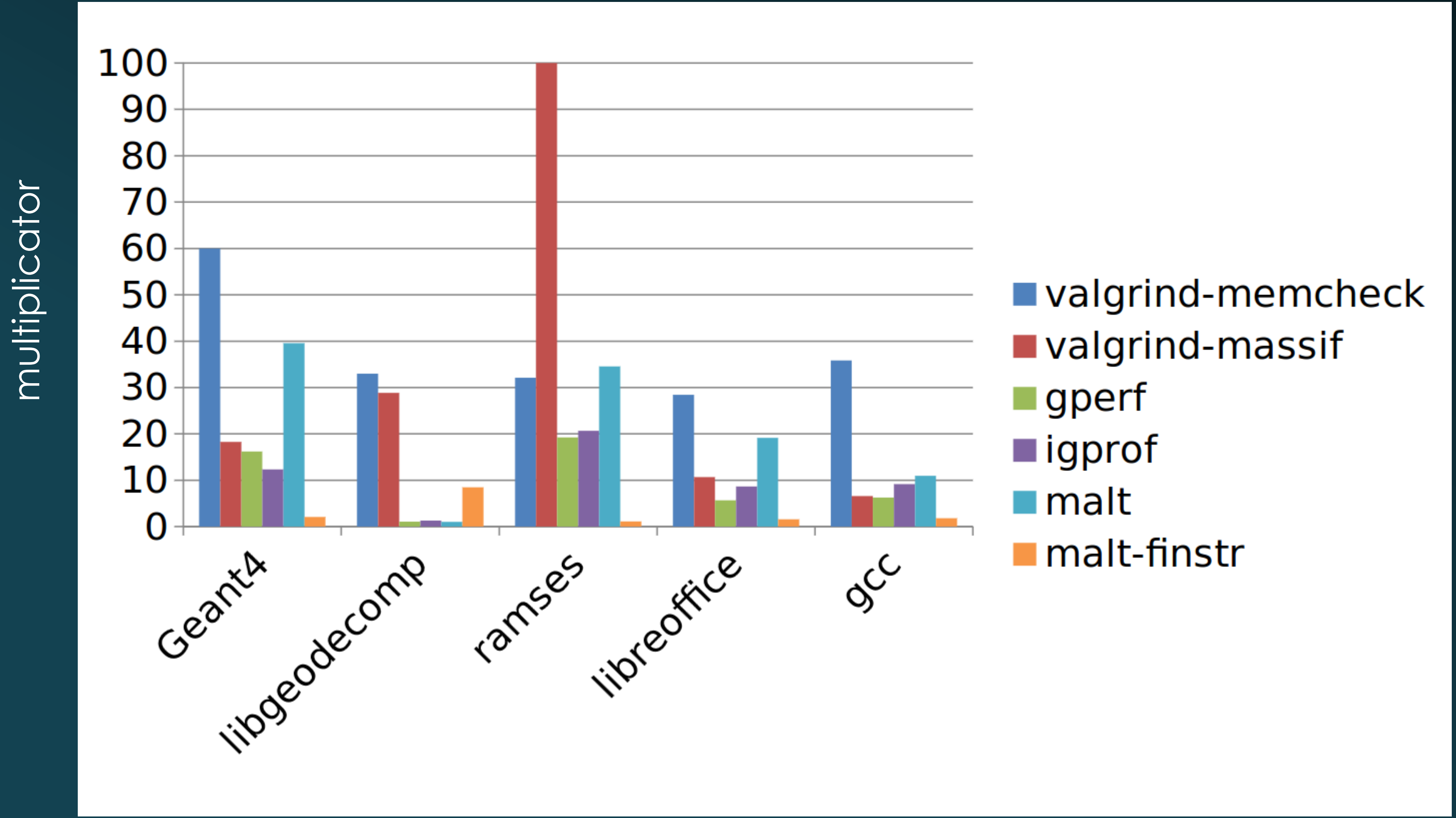


Search for the minimal chunk size.

Many codes produce allocations of 1B. OK with moderation.

```
      case (DATATYPE_REAL_NODE_VECTOR,DATATYPE_REAL_ELEM_VECTOR, &
            DATATYPE_REAL_FACE_VECTOR,DATATYPE_REAL_PAIR_VECTOR)
         if (associated(data_ptr%r2_ptrs)) then
            deallocate(data_ptr%r2_ptrs)
         end if
1 B      allocate(data_ptr%r2_ptrs(nel_grps))
         do n=1,nel_grps
            NULLIFY(data_ptr%r2_ptrs(n)%ptr)
         end do
```

Overhead

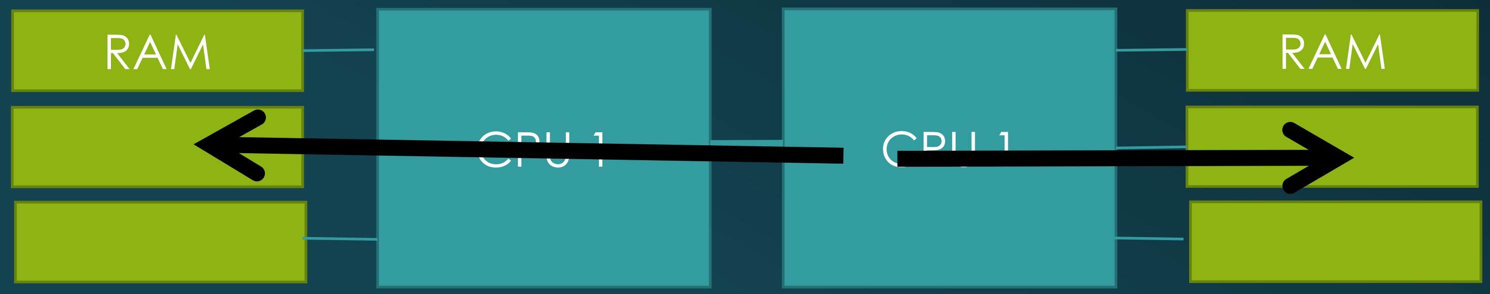

multiplicator
100
90
80
70
60
50
40
30
20
10
0
Geant4
libgeodecomp
ramses
libreoffice
gcc
valgrind-memcheck
valgrind-massif
gperf
igprof
malt
malt-finstr

Reminder on NUMA

Non Uniform Memory Access

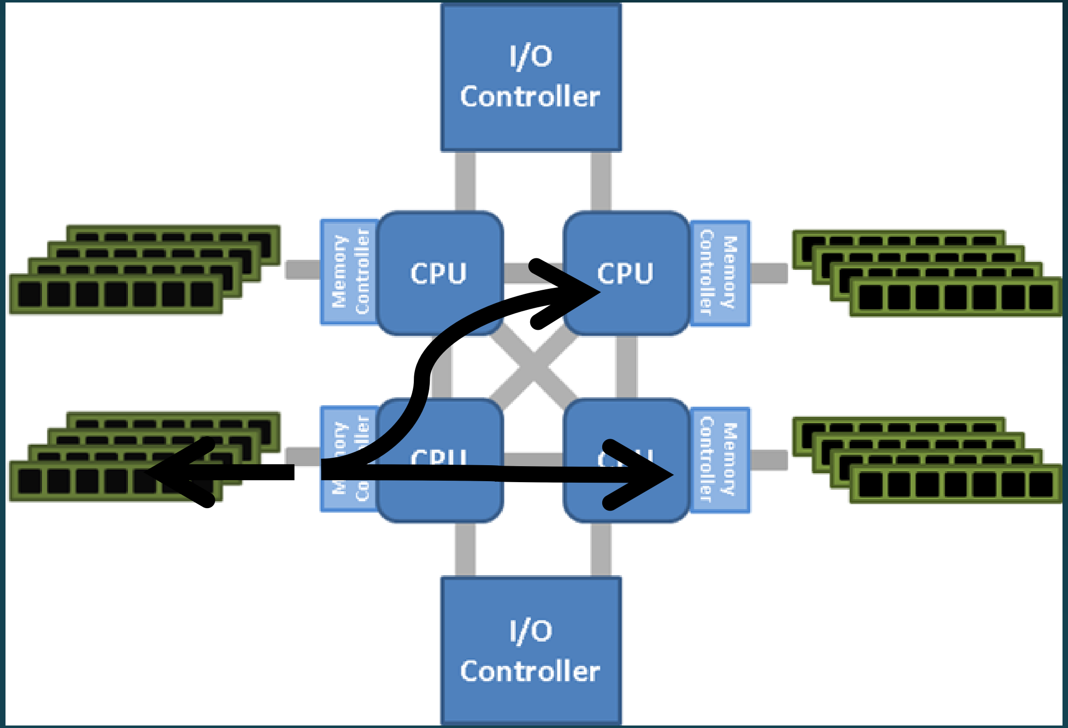
What is NUMA ?

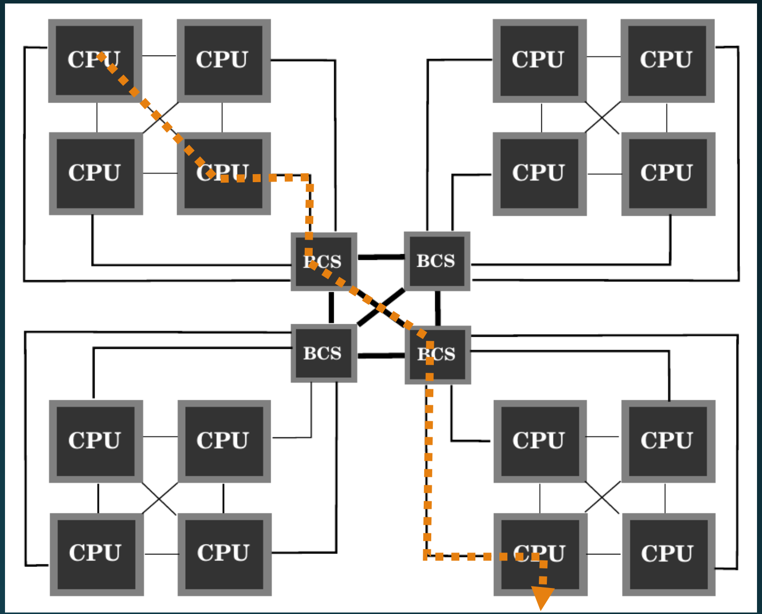
- Each CPU has its **own memory**
- Access to **remote memory** we need to **go through the owner** CPU



Architecture

- **A bit more complex**

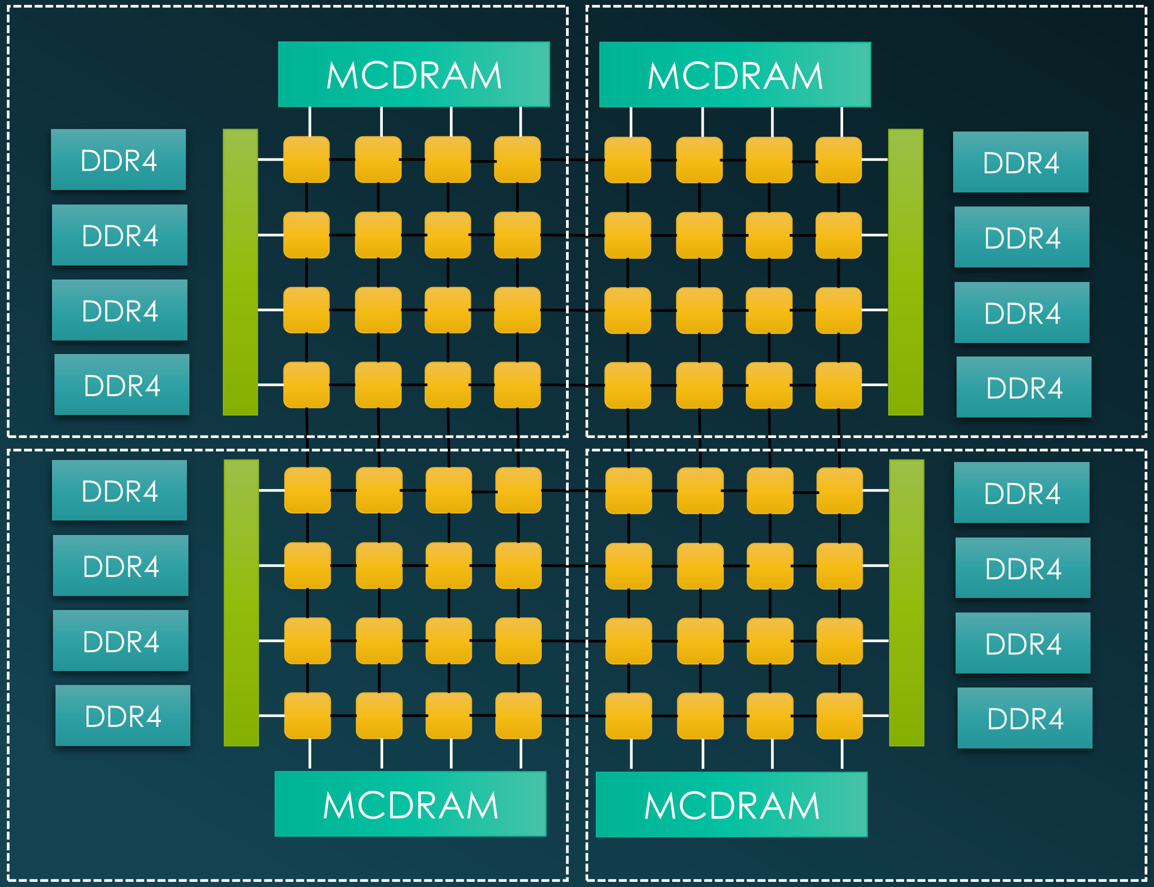




Large nodes : 16 proc - 128 cores
Bull BCS ~ 2010

Today topology

- **Intel Knight Landing,** mode **SNC2** or **SNC4**
- Also add fast memory **MCDRAM (HBM)** presented as **NUMA** or **LLC cache**

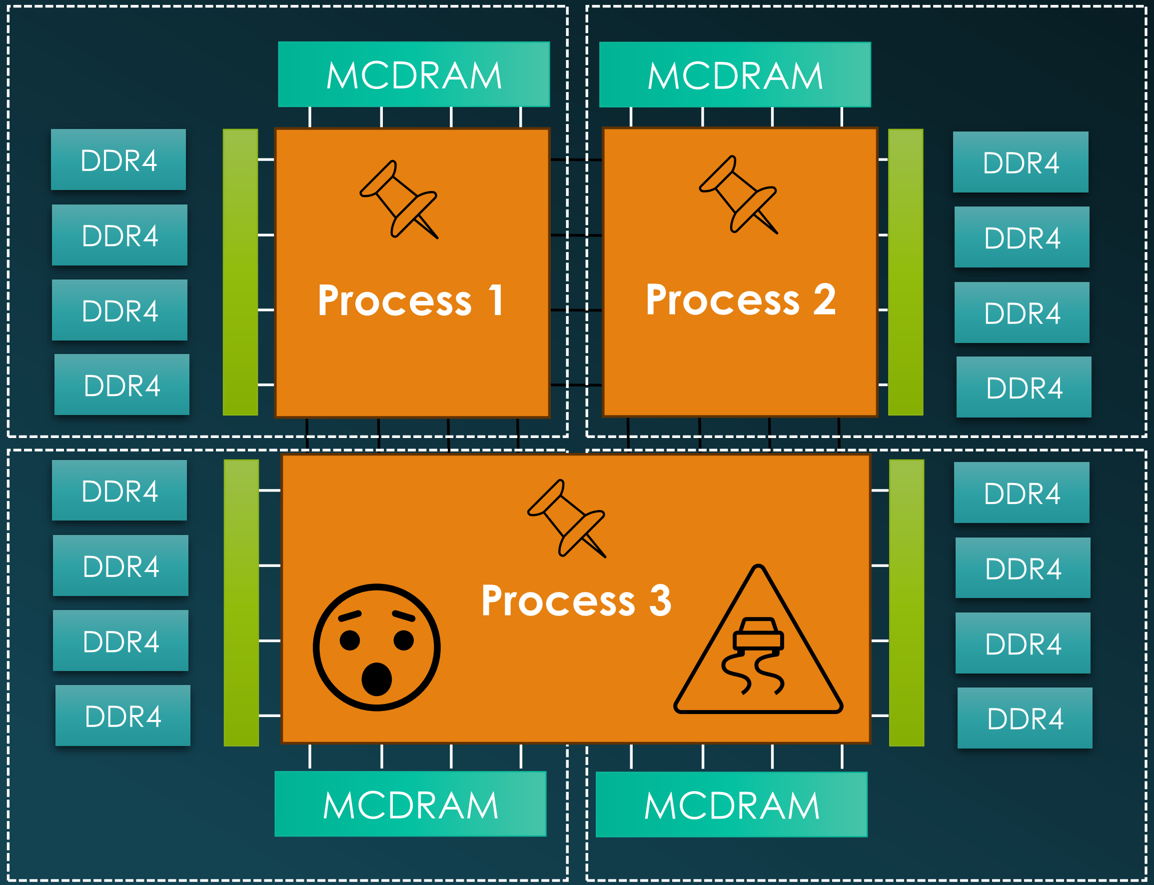


Today topology

Safe usage :

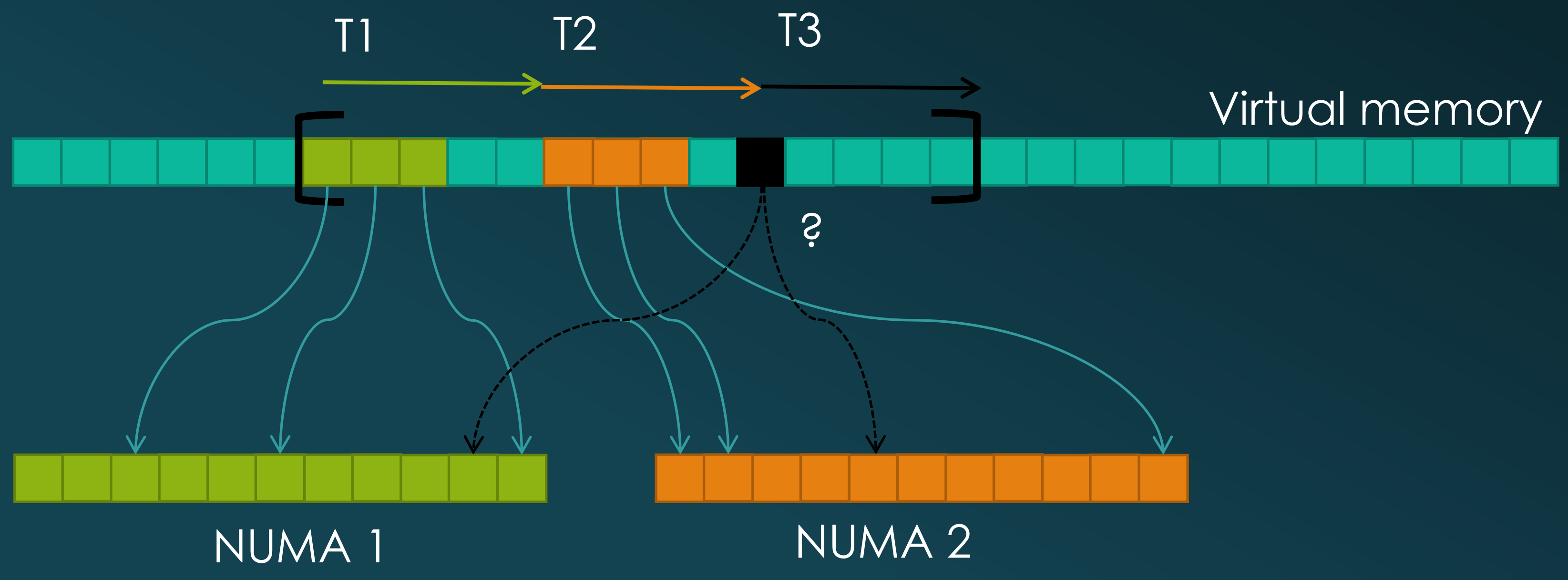
Each **process** bound over

a **single NUMA node**



Implicit binding : first touch

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



Typical OpenMP mistake

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

```
memset(array, 0, SIZE)
```

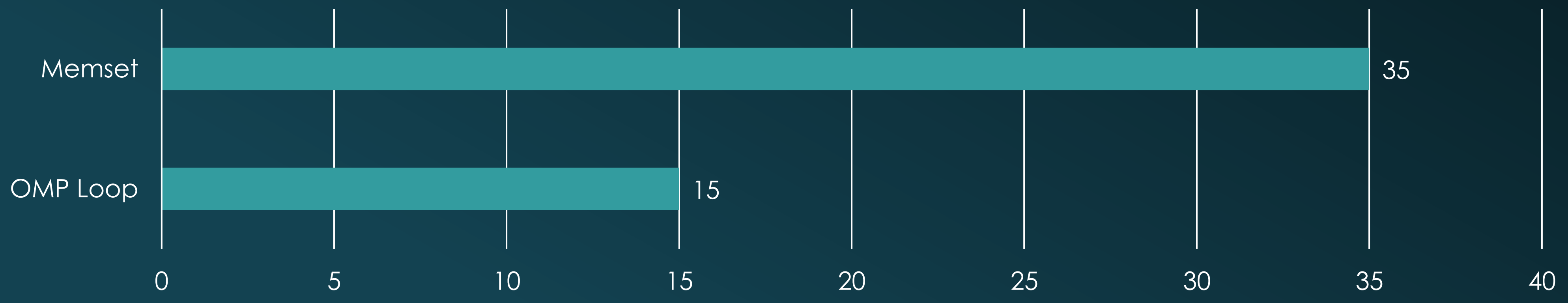
- Then access

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

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

Performance impact

- On a real machine (2 NUMA) :



Wish list for a profiling tool…

- 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**

https://www.freepng.fr/png-nmoqvw/

NumaProf

How to know if we are right In a real application ?

With the help of **Othman Bouizi**

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
  - **Difficulty**: we cannot easily use libs inside the tool
  - I would have used hwloc and libnuma…..

On access we need…

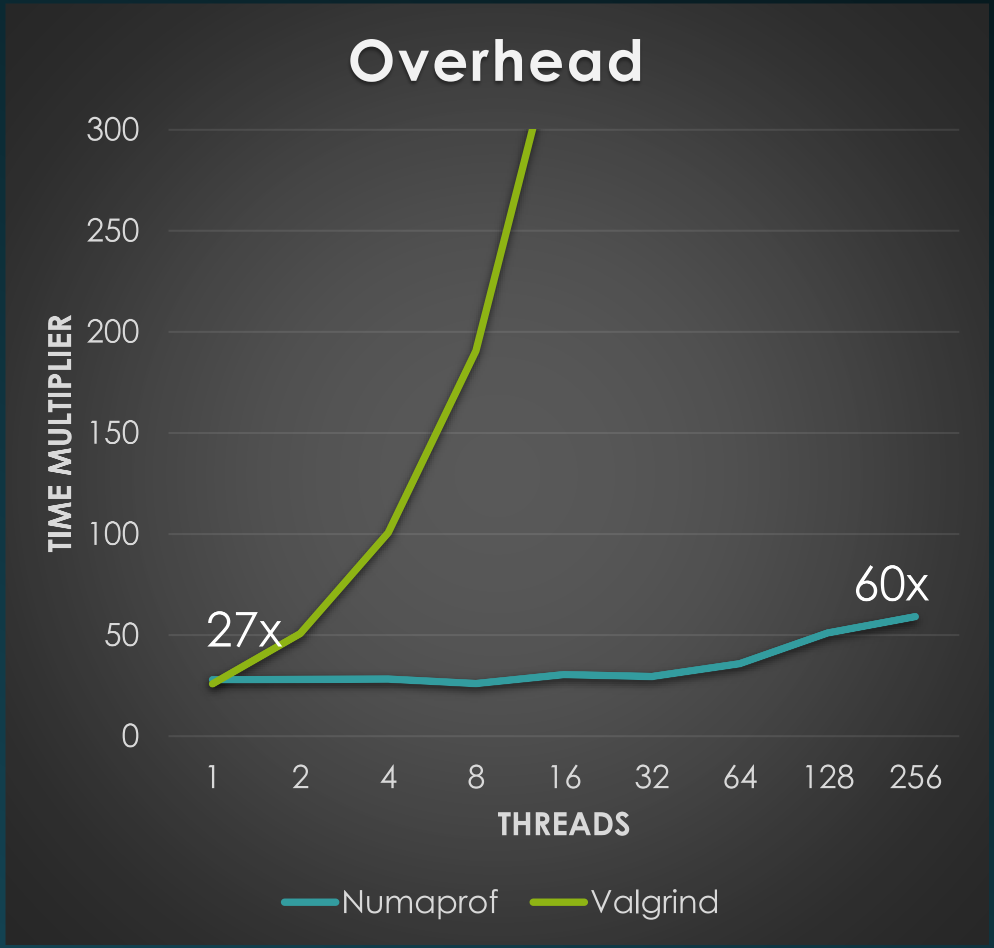
- On each access we want to know if it is
  - **Remote** access
  - **Local** access
  - **MCDRAM** access
  - **Page is pinned**
  - **Thread is pinned**

- So, we need to know
  - Where is **the page**
  - Where is **the current thread**

- We can **skip** accesses to **local stack** (overhead 80x -> 40x)

Overhead and scalability

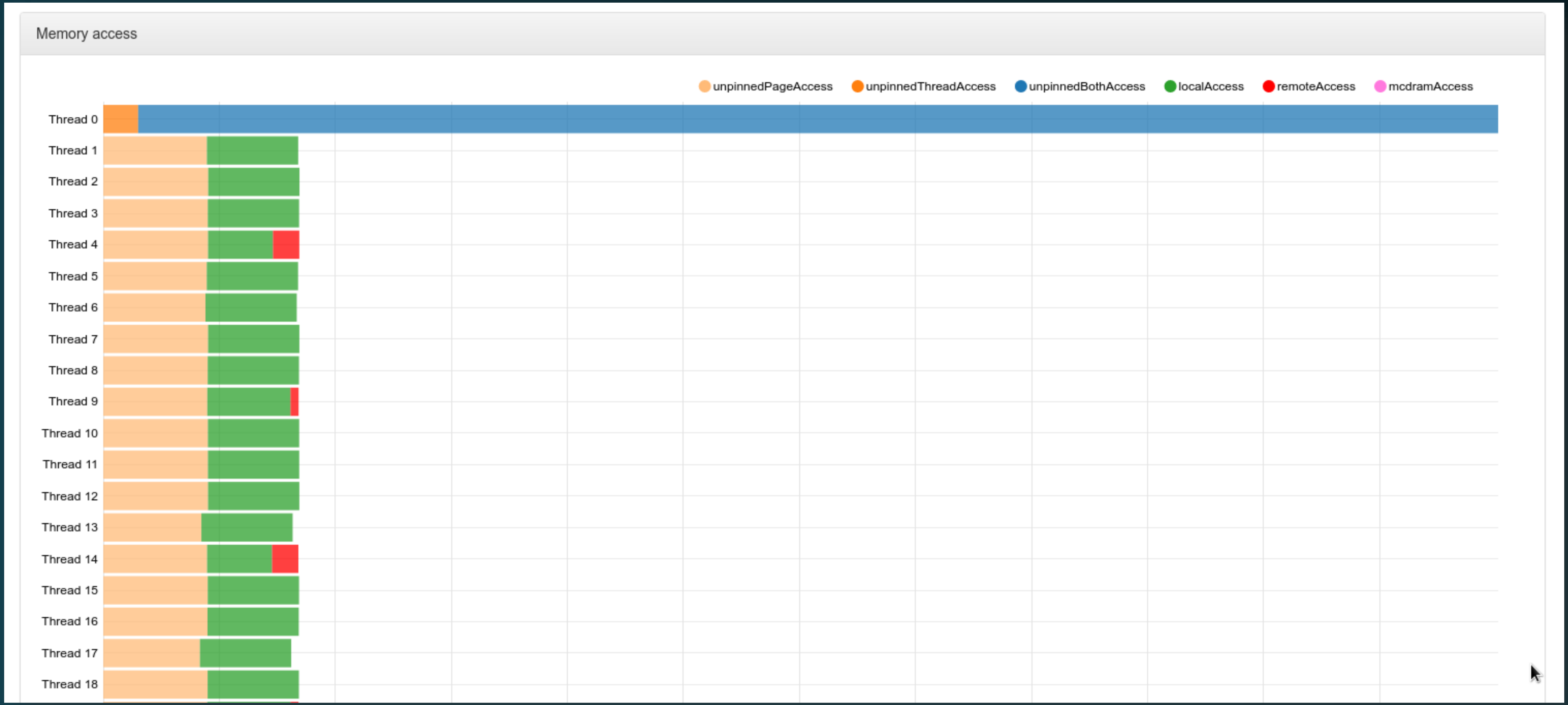
- Of course overhead is large: **~30x**
- But is **scale**
- Example code hydro on **KNL**:



GUI and example

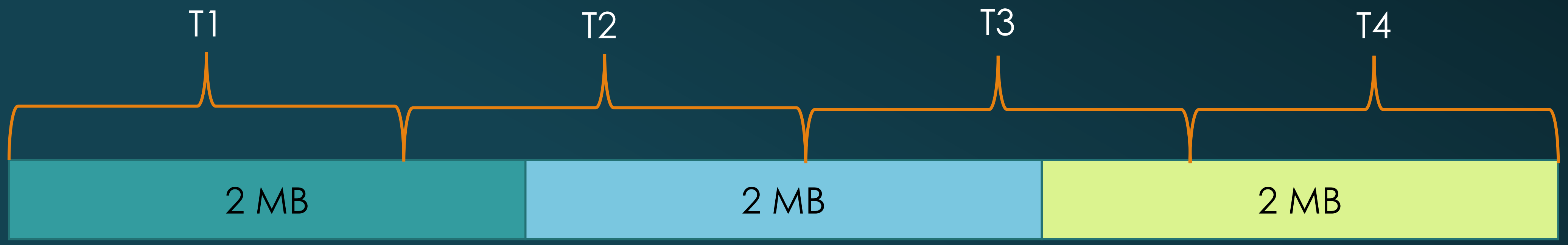
Global summary

Statistics per thread

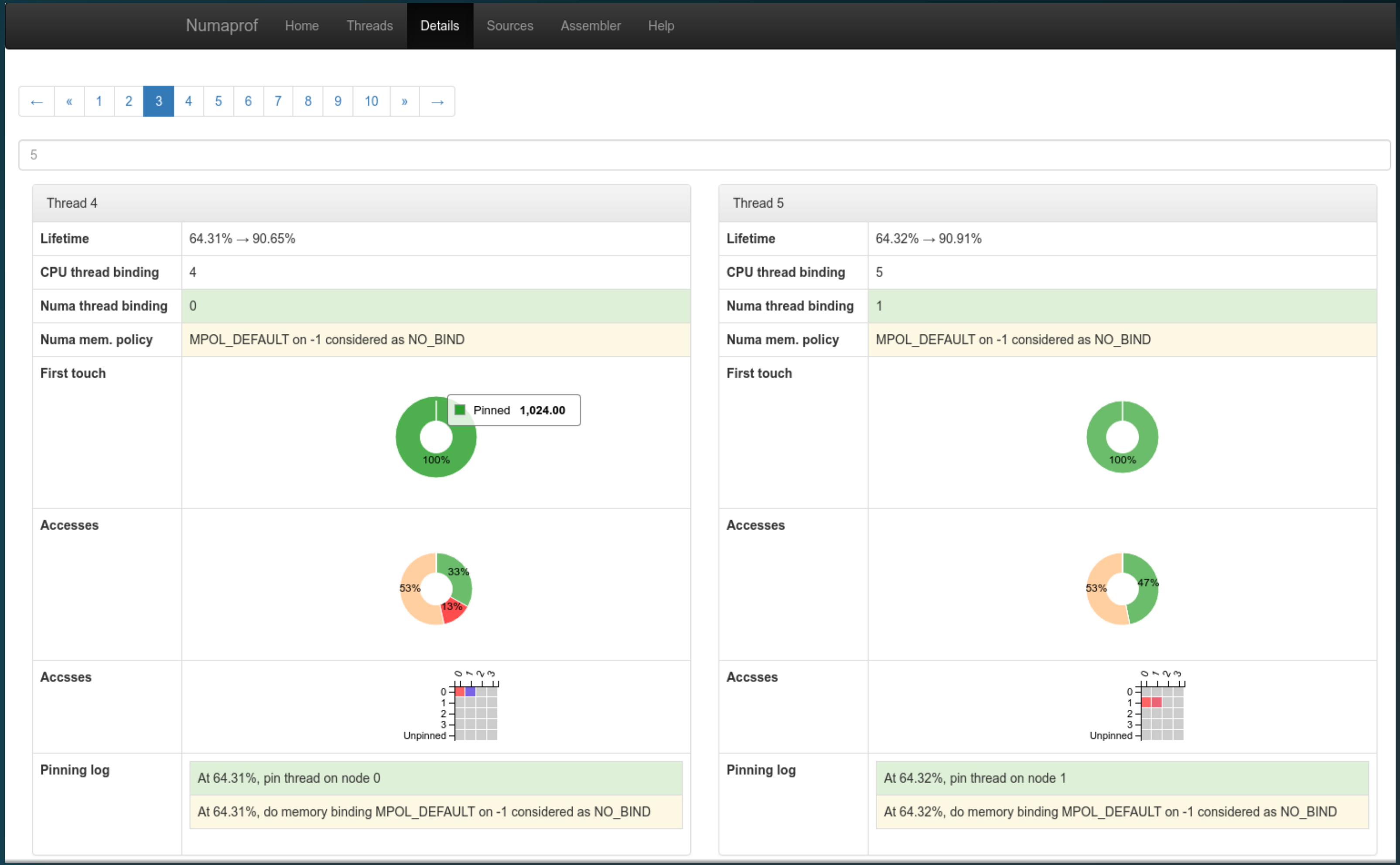


OMP and huge pages

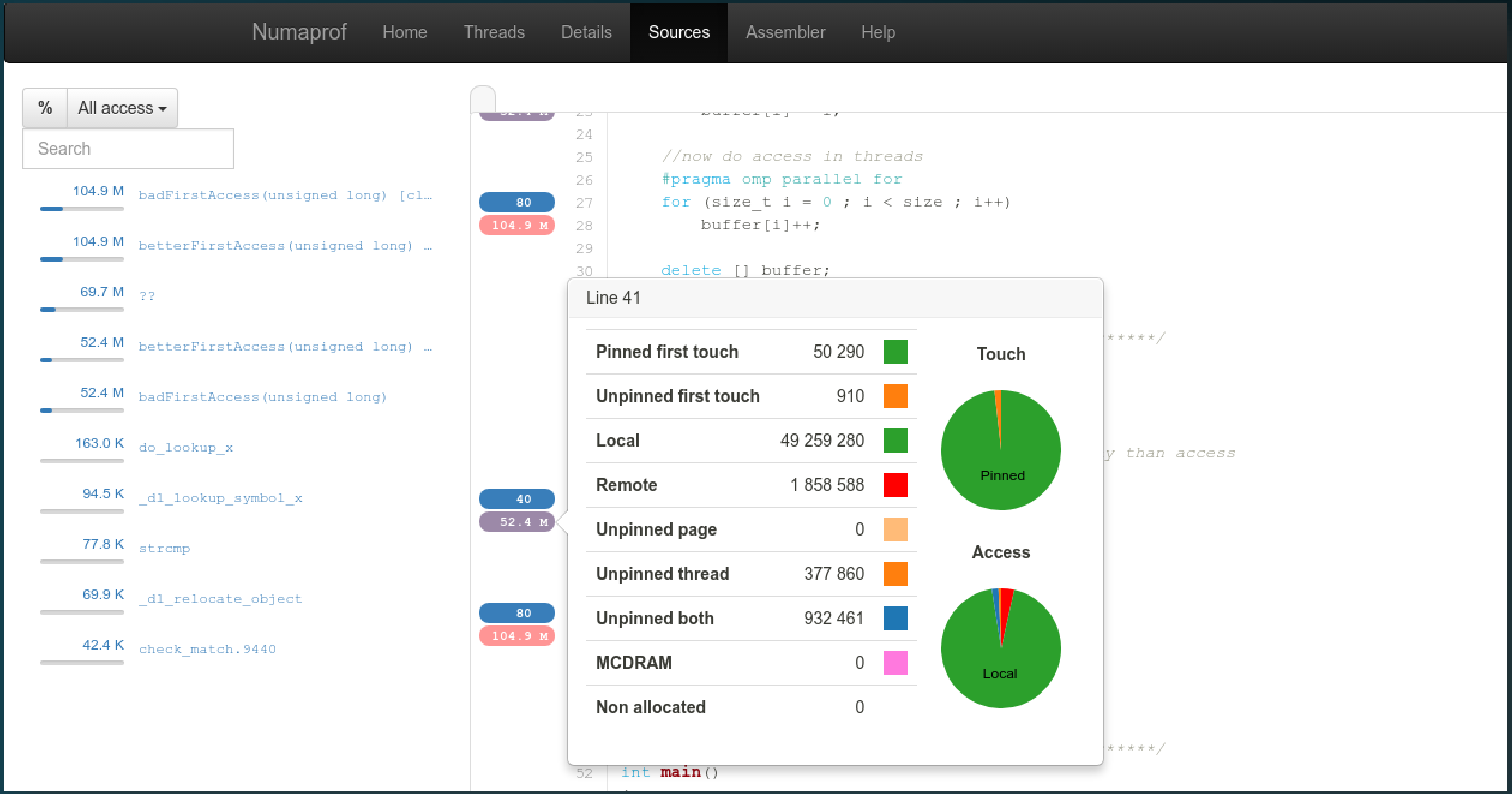
- **Huge pages** & thread splitting
- Most of the time do not match exactly
- Not a big issue if limited



Details per thread



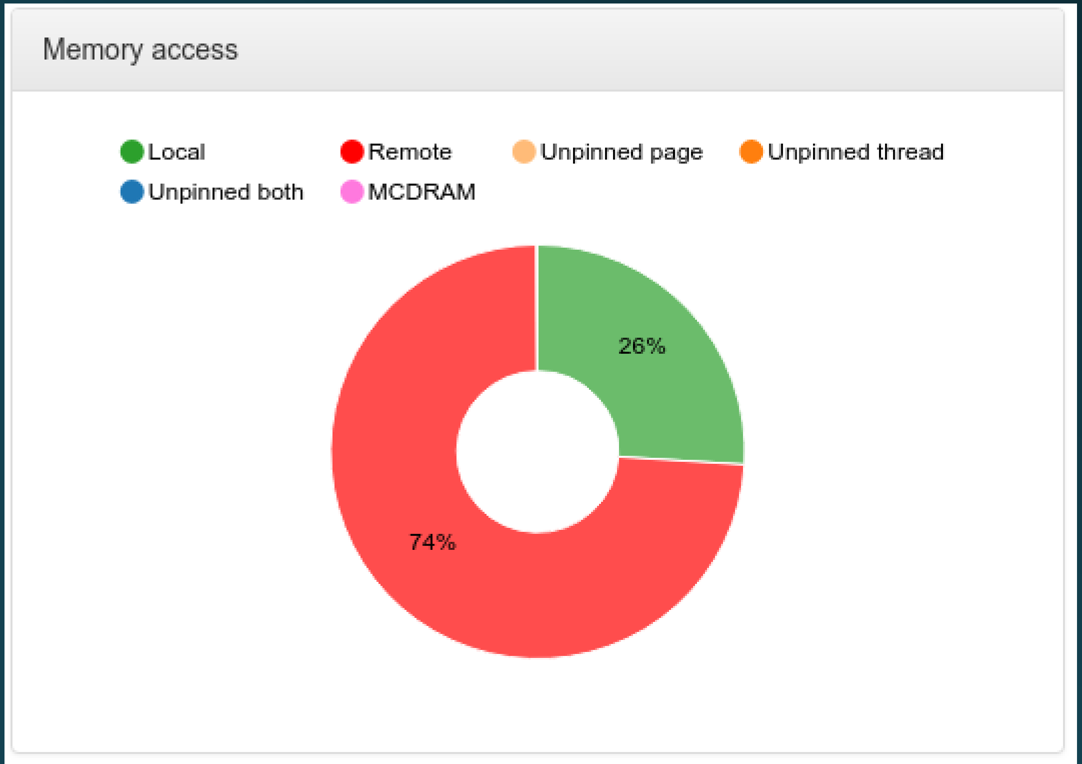
Source & asm annotations

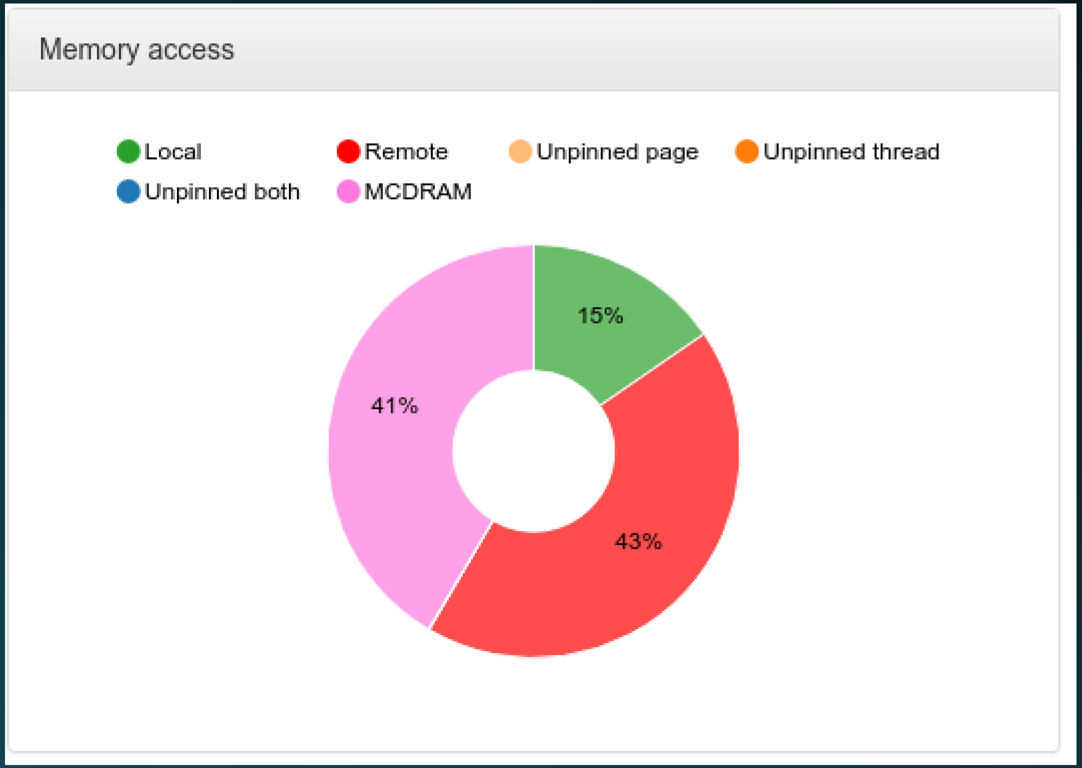


Code Hydro

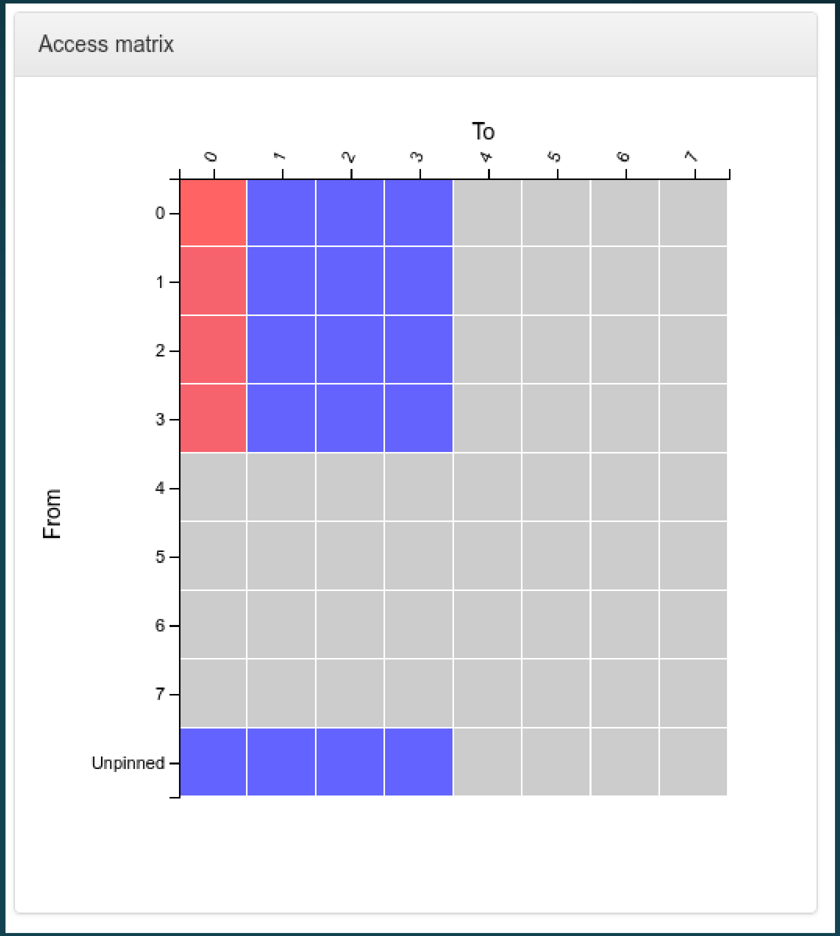
- KNL Without HBM

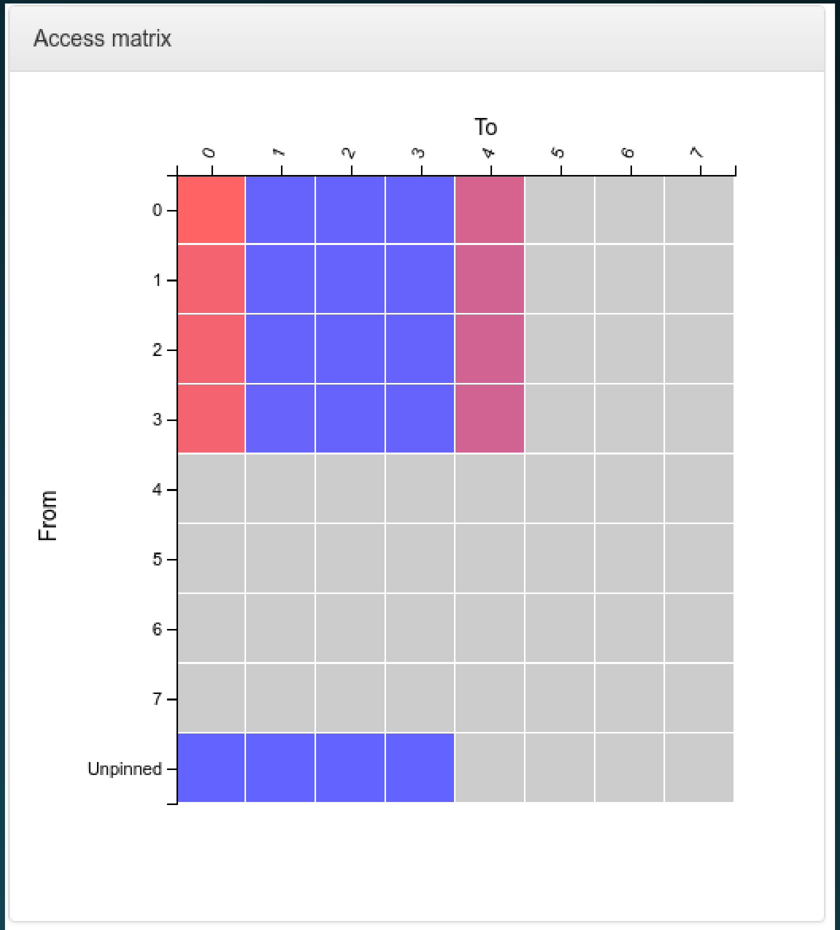
WITH HBM



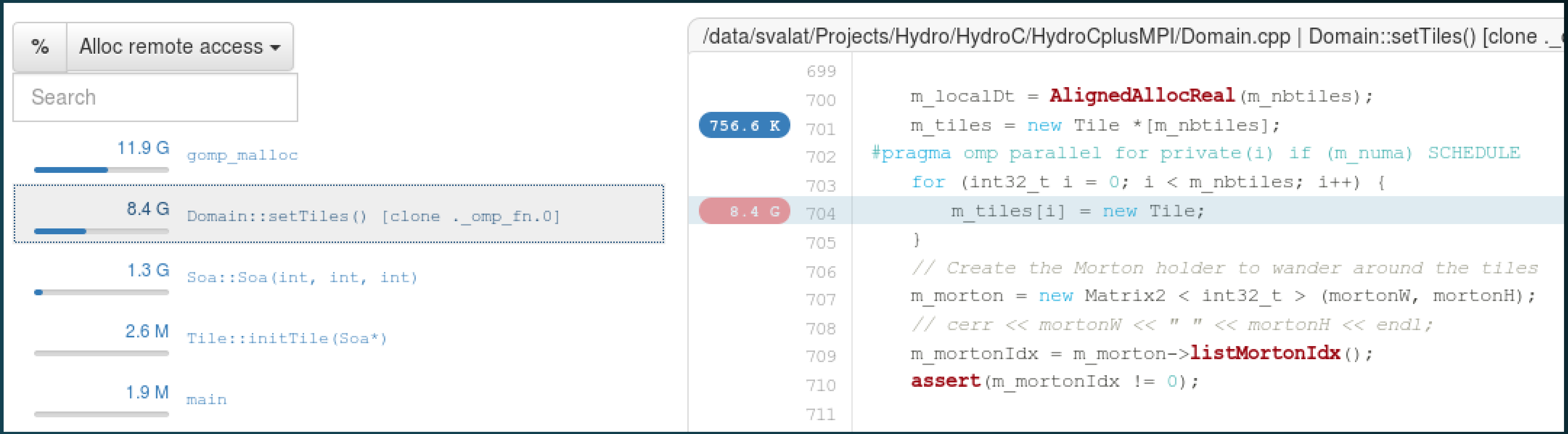


Original Hydro access matrix



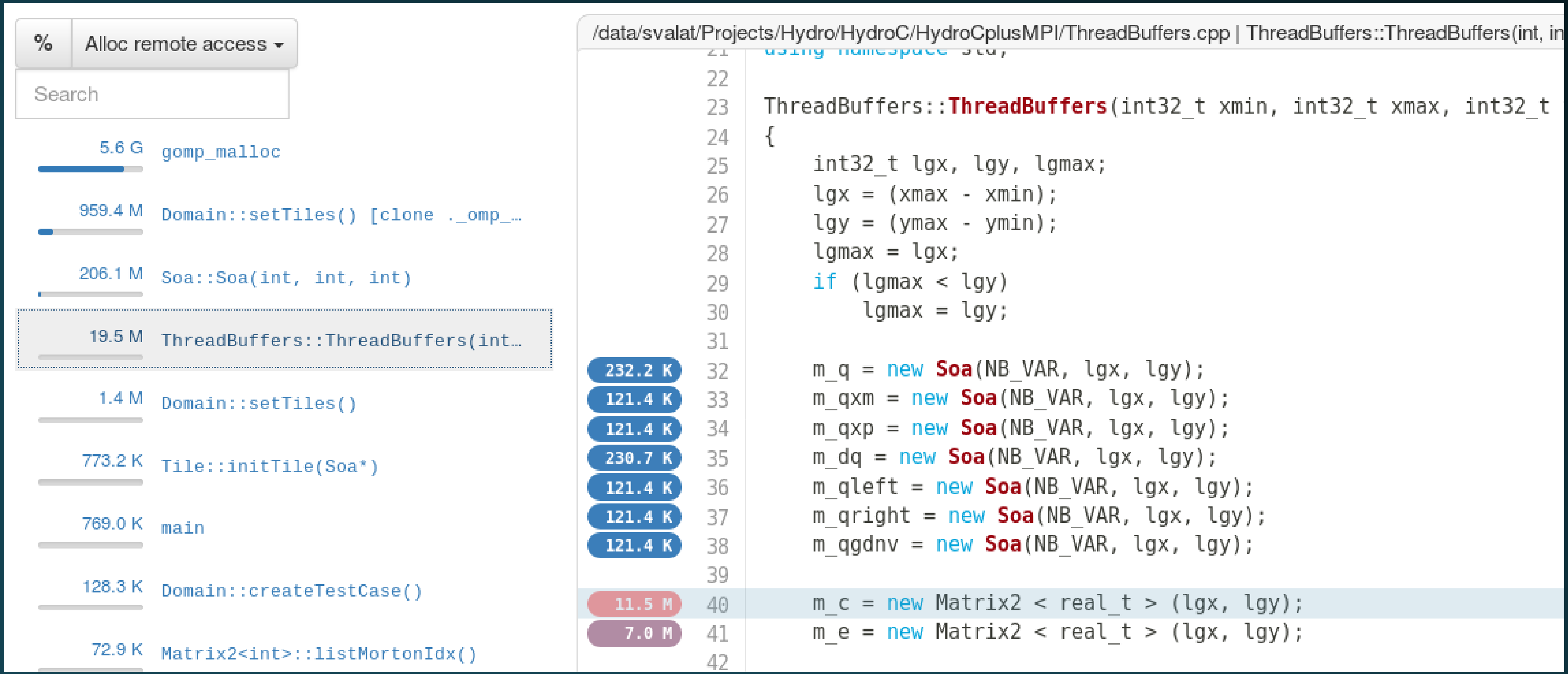


Ordering issue




```
#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;
        }
```

Non parallel allocations



Parallel allocations

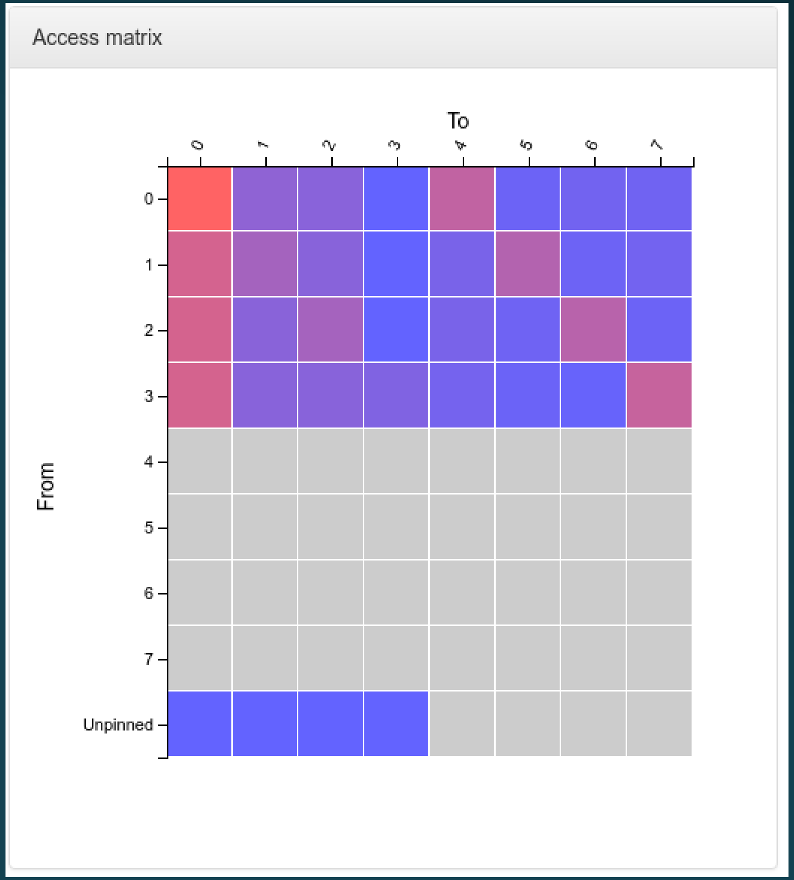
- Original

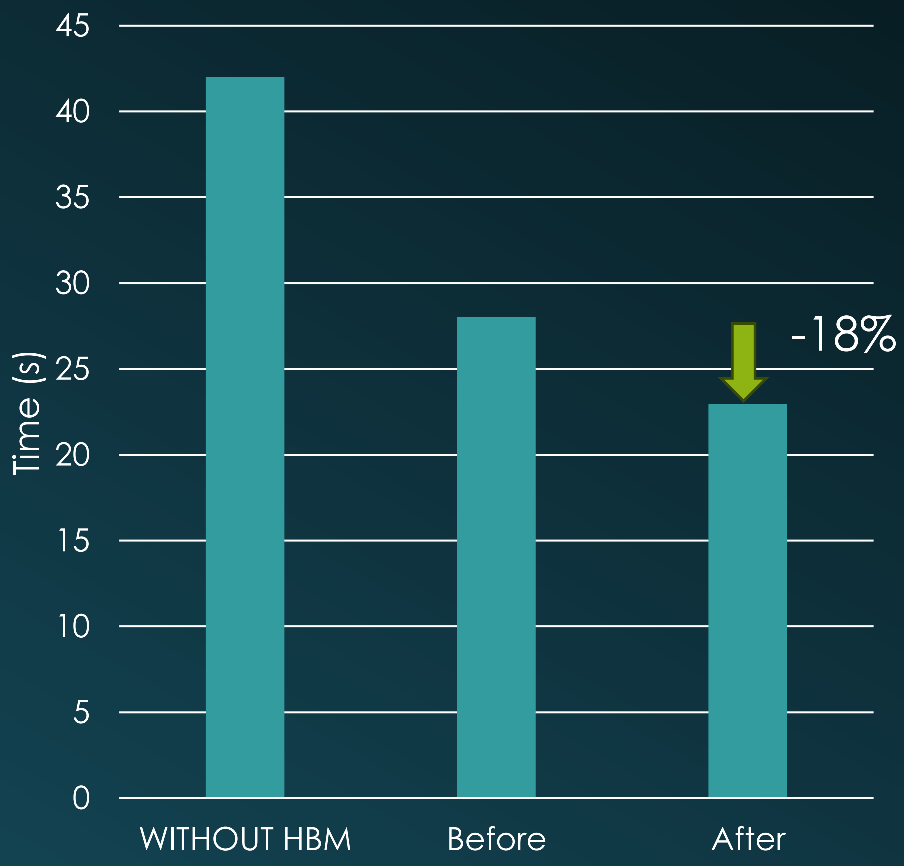
```
for (int32_t i = 0; i < m_numThreads; i++) {
	m_buffers[i] = new ThreadBuffers(...);
	assert(m_buffers[i] != 0);
}
```

- 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





Conclusion

- **Memory** is **not trivial** to handle **in large programs**

- Need to be taken in account

- Some **tiny mistakes** sometimes **cost a lot**
  - **Possibly everywhere** in the program (**global impact**)

- Be able to get a view is a first help

Questions ?

MALT
A Malloc Tracker

NUMAPROF
A NUMA profiler

https://memtt.github.io

SCAN ME