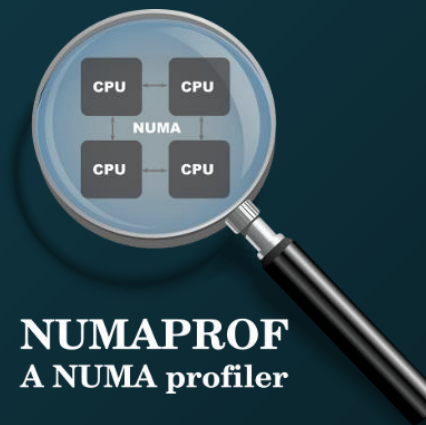




**MALT**  
A Malloc Tracker



**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

The screenshot shows the Vi-HPS website header with the logo and navigation links: MASTODON, TWITTER/X, IMPRINT, PRIVACY, and SITEMAP. The main navigation bar includes ABOUT, MEMBERS, ORGANIZATION, TOOLS (selected), TRAINING, SYMPOSIA, PROJECTS, and NEWS. The page title is 'VIRTUAL INSTITUTE — HIGH PRODUCTIVITY SUPERCOMPUTING'. The main content area is titled 'TOOLS OVERVIEW' and contains the following text:

To assist developers with the selection of the appropriate tools provided by VI-HPS, our Tools Guide booklet linked below offers a brief overview of the respective tools. The guide showcases their individual debugging, correctness checking and performance analysis capabilities. Furthermore, it indicates their support for parallel computer systems, programming models and languages.

- [Vi-HPS Tools Guide](#)

Detailed information about each particular tool can be found on the listed websites in the document above and the respective tool pages below.

<p>Single Node Performance</p> <ul style="list-style-type: none"> <li><a href="#">Callgrind</a></li> <li><a href="#">IKWID</a></li> <li><a href="#">MAQAO</a></li> </ul>	<p>Instrumentation</p> <ul style="list-style-type: none"> <li><a href="#">Qpari2</a></li> </ul>
<p>Parallel Performance</p> <ul style="list-style-type: none"> <li><a href="#">Dimemas</a></li> <li><a href="#">Extra-P</a></li> <li><a href="#">Linaro MAP (Part of Linaro Forge)</a></li> <li><a href="#">Linaro Performance Reports (Part of Linaro Forge)</a></li> <li><a href="#">mpiP</a></li> <li><a href="#">Open SpeedShop</a></li> <li><a href="#">Paraver</a></li> <li><a href="#">Scalasca</a></li> <li><a href="#">TAU</a></li> <li><a href="#">Vampir</a></li> </ul>	<p>Measurement</p> <ul style="list-style-type: none"> <li><a href="#">Extrac</a></li> <li><a href="#">PAPI</a></li> <li><a href="#">Score-P</a></li> </ul> <p>Integration</p> <ul style="list-style-type: none"> <li><a href="#">Component-based Tool Framework</a></li> <li><a href="#">LaunchMON</a></li> <li><a href="#">pMMPi</a></li> </ul> <p>Visualization</p> <ul style="list-style-type: none"> <li><a href="#">Cube</a></li> </ul>

# Origin of the tools

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



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



- **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€

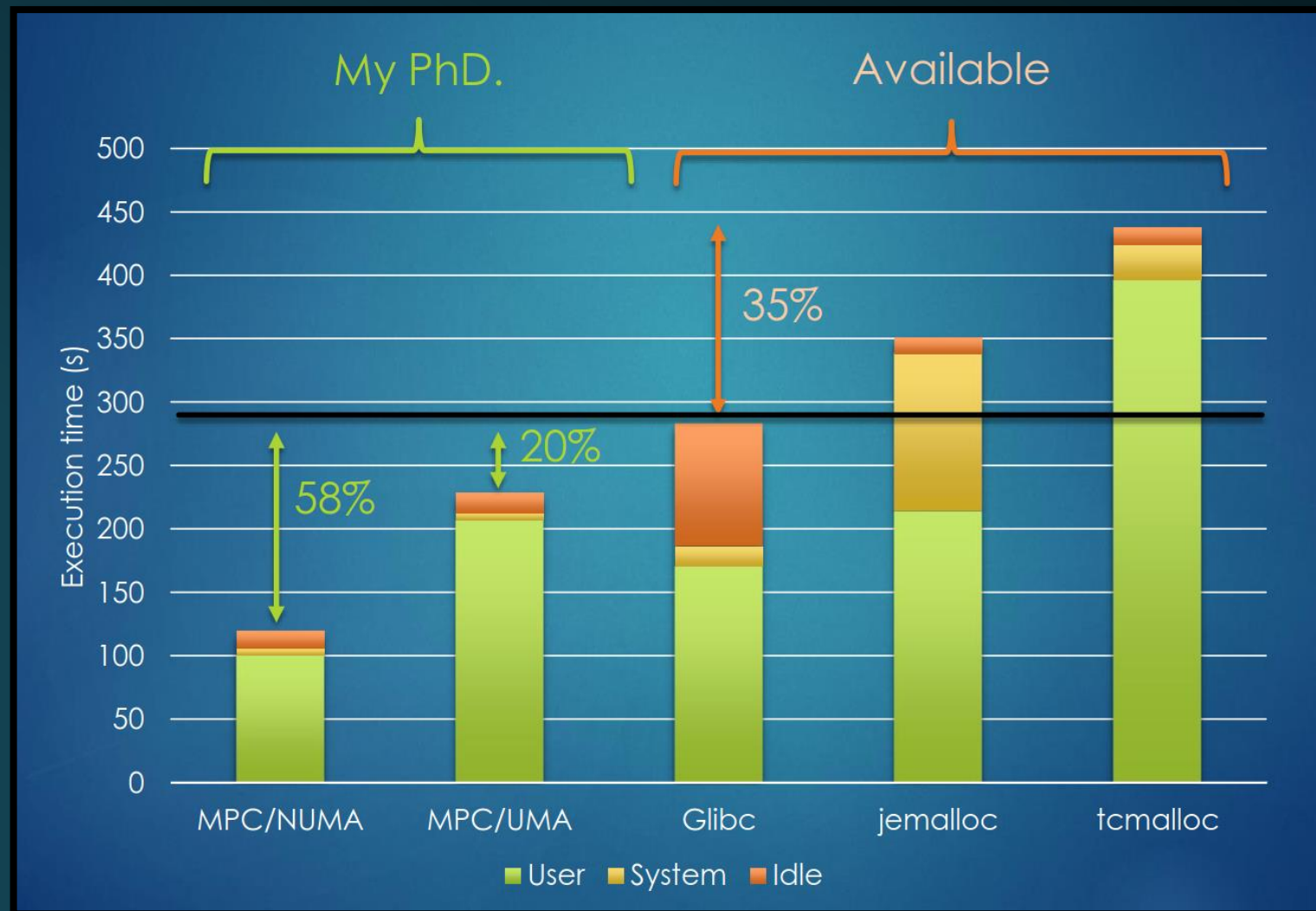


# Key today

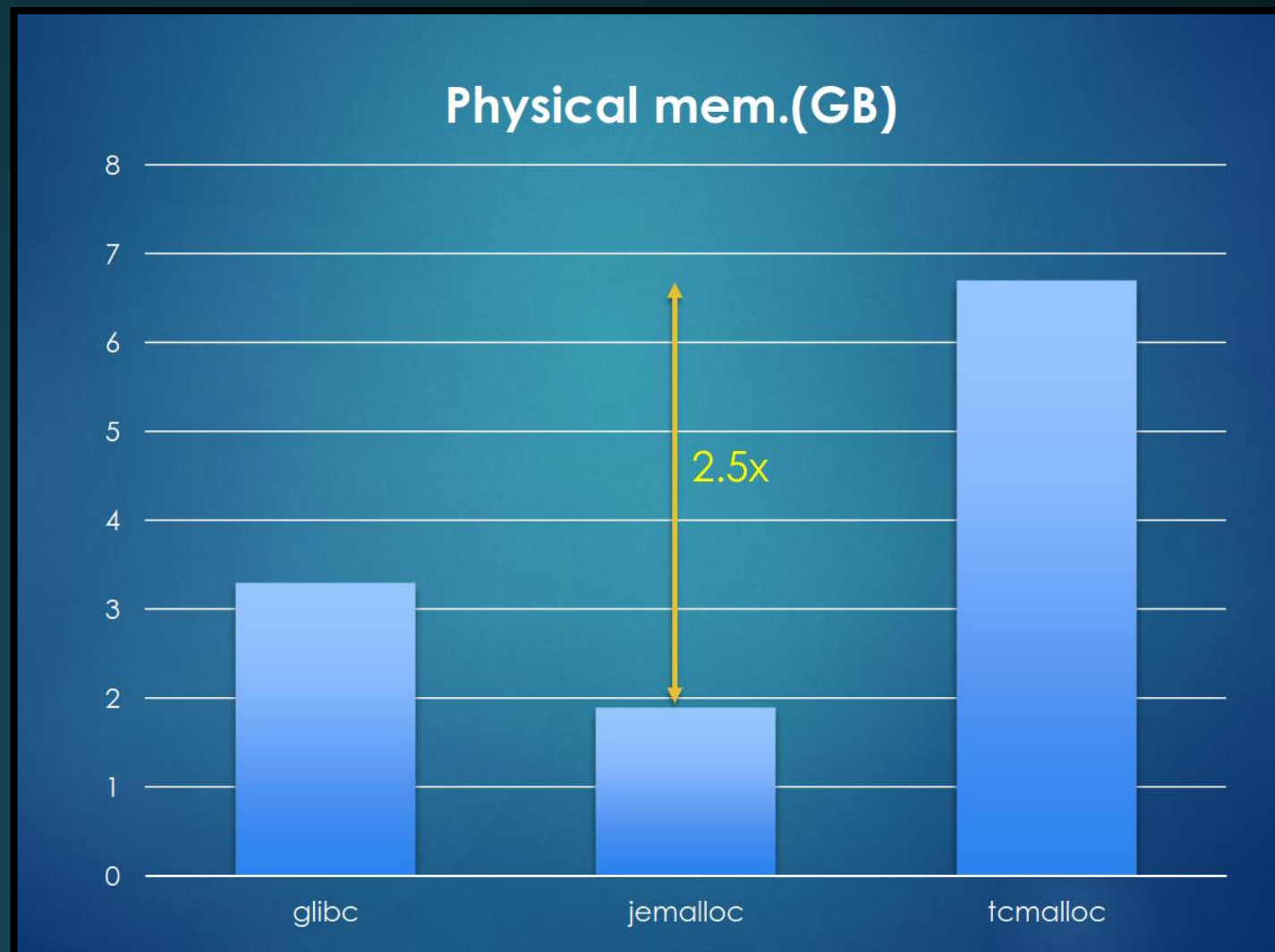
You need to  
**well understand**  
**the memory behavior**  
of your (HPC)  
application !



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



# Same about **memory consumption** on 12 cores





# MALT

Malloc Tracker



**MALT**  
A 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 globalVar[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**

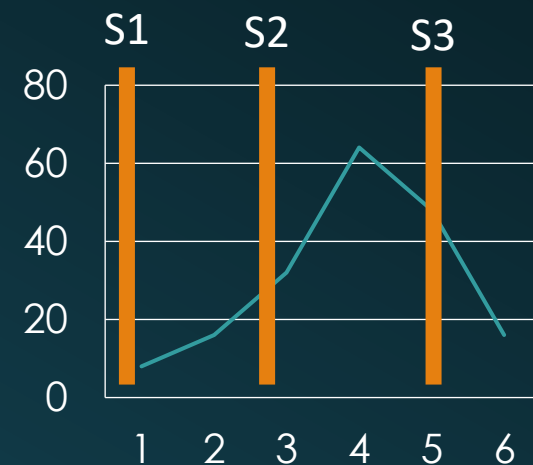
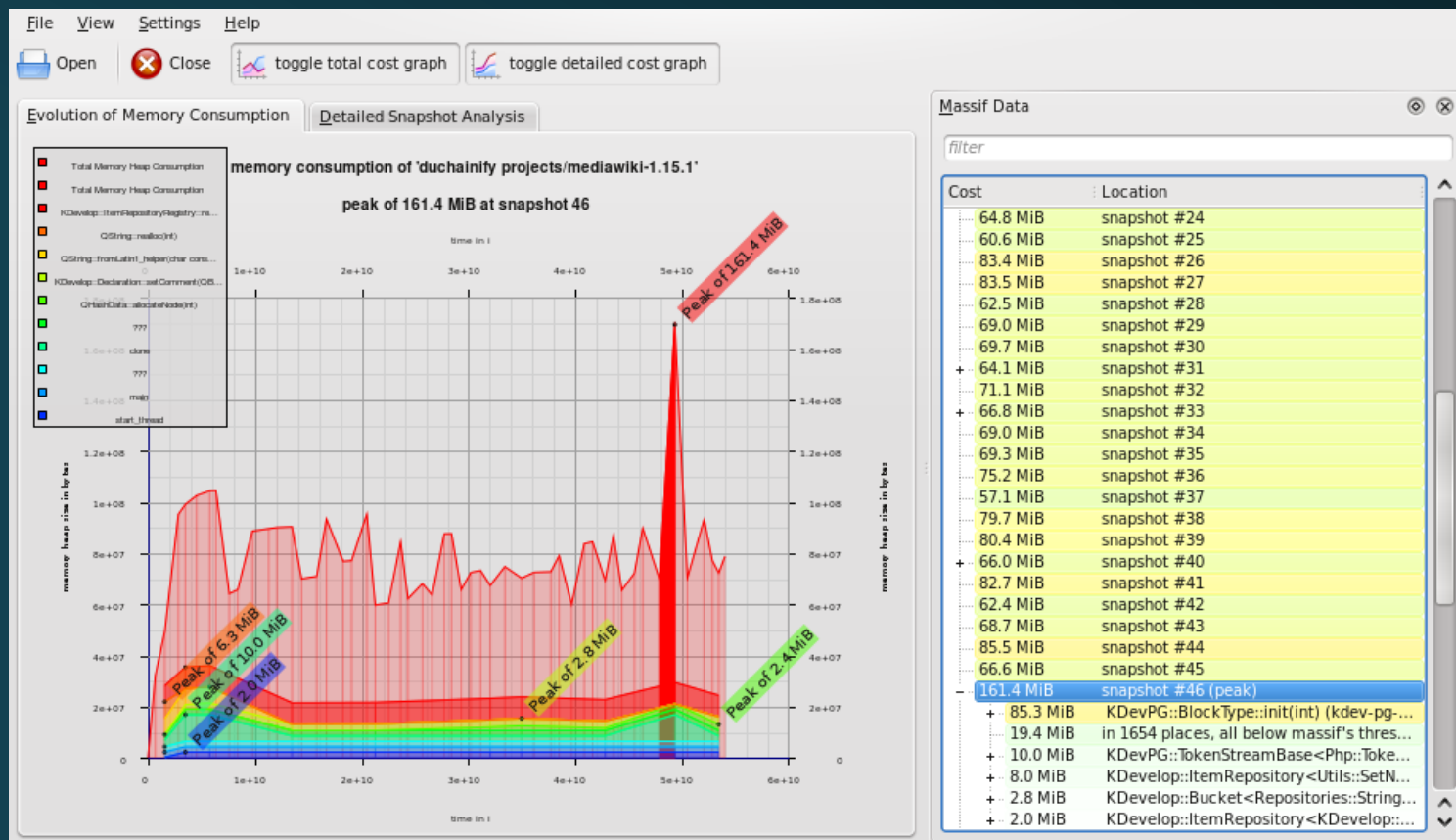
Might lead to swap for **large size**

**Short life** allocations



# Existing tools

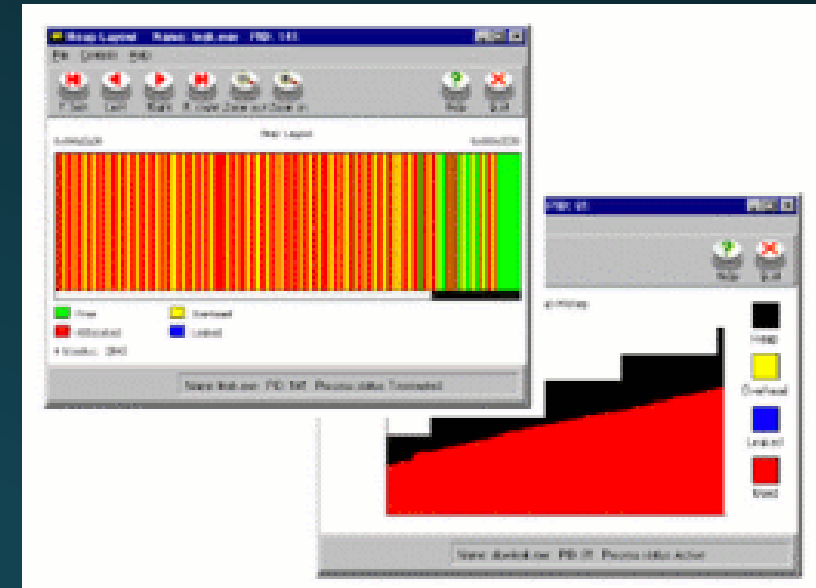
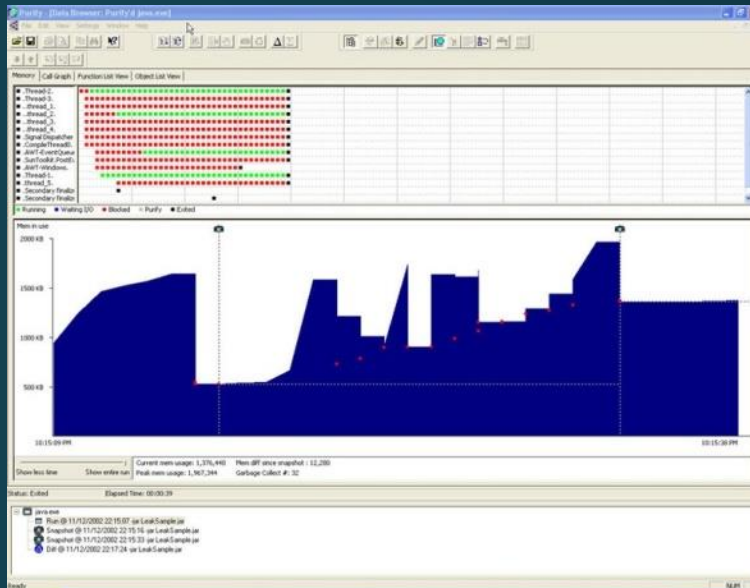
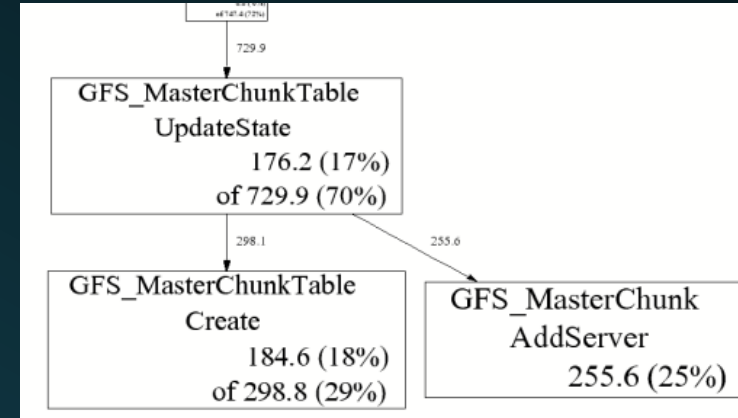
- Valgrind - **memcheck**
- Valgrind - **massif**



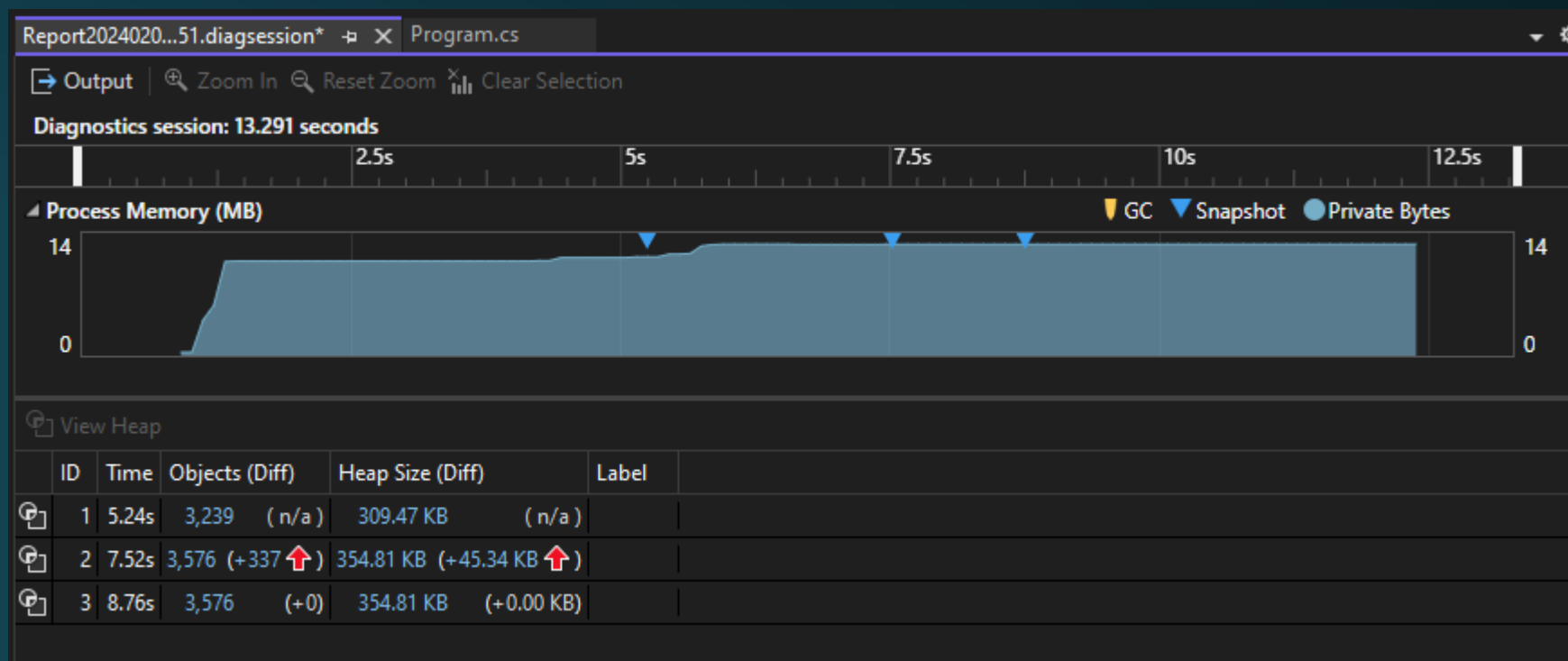
# Existing tools

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

➤ **IBM Purify++** / **Parasoft Insure++**



# Visual Studio Ultimate memory profiler



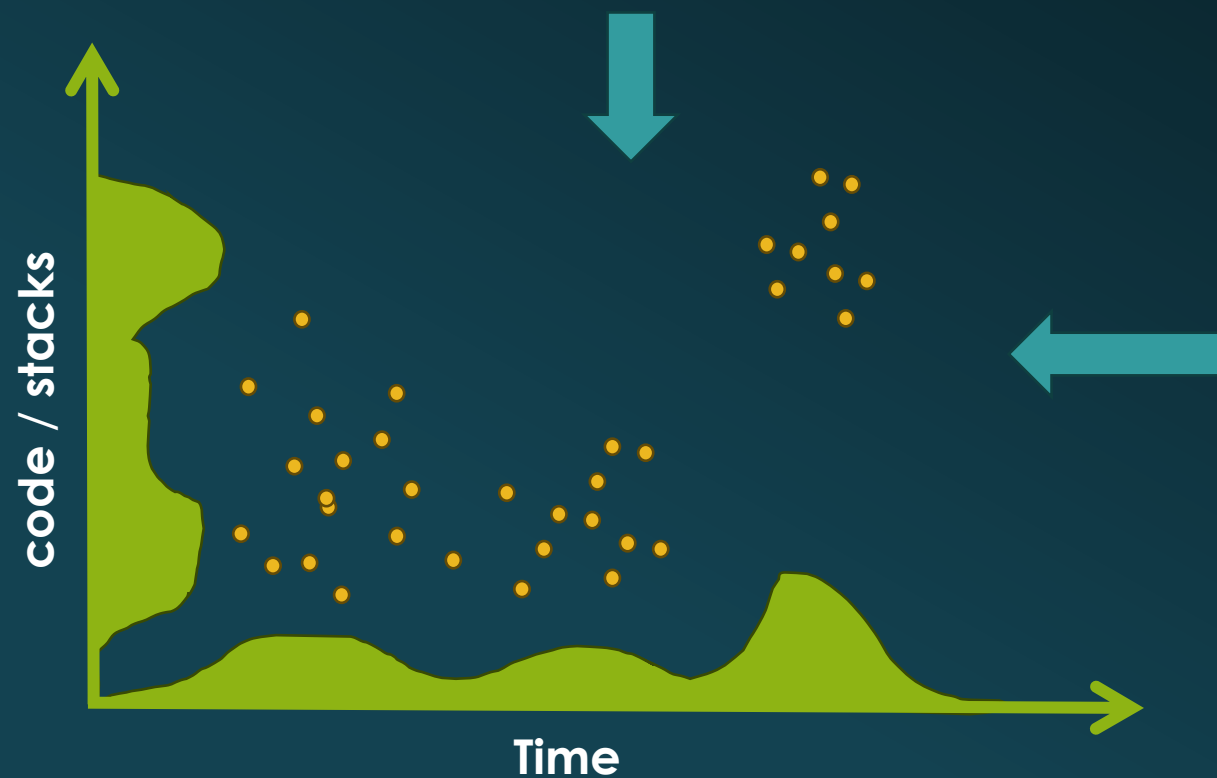
# What I want to provide

- 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

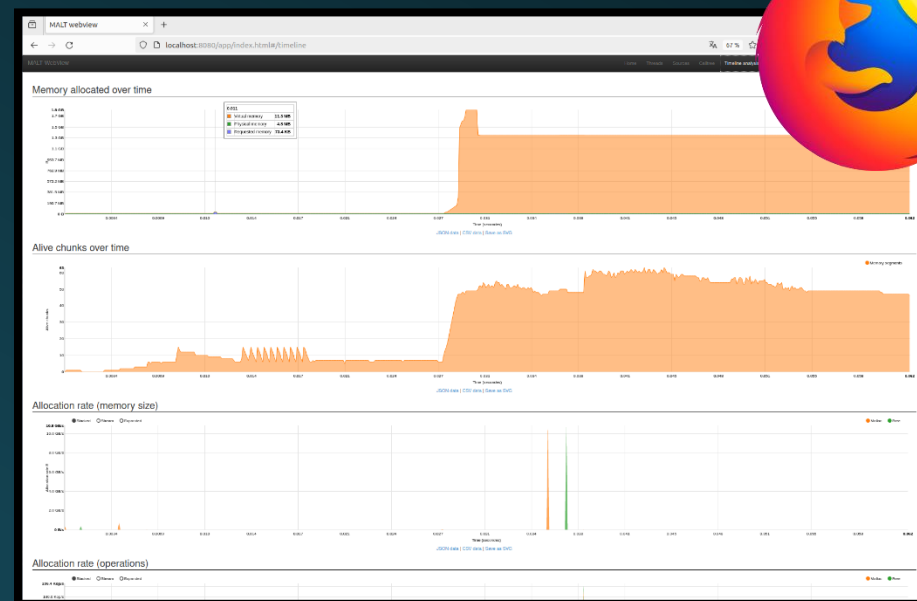
## Viewing via light web server

```
malt-webview ..
```

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

Can be forwarded via SSH

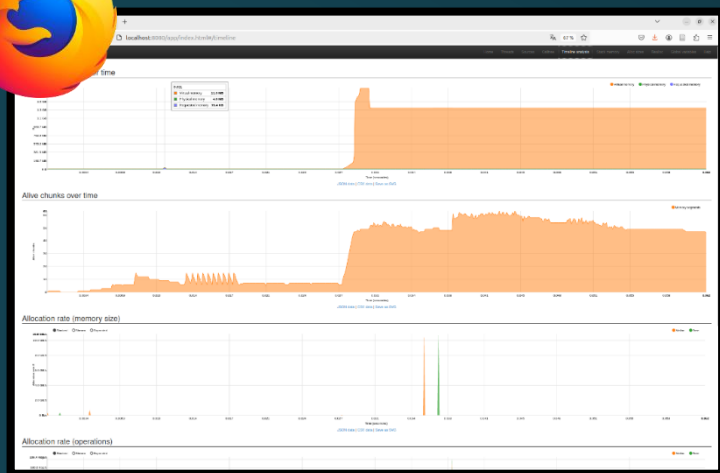
```
ssh -L8080:localhost:8080 .....
```



# 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  
00:00:00.25

PHYSICAL MEMORY PEAK  
2.3 MB

ALLOCATION COUNT  
379

AVAILABLE PHYSICAL MEMORY  
4.1 Gb

## Run description

Executable :	simple-case-finstr-linked
Commande :	<code>./simple-case-finstr-linked</code>
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	2.0 KB



# 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

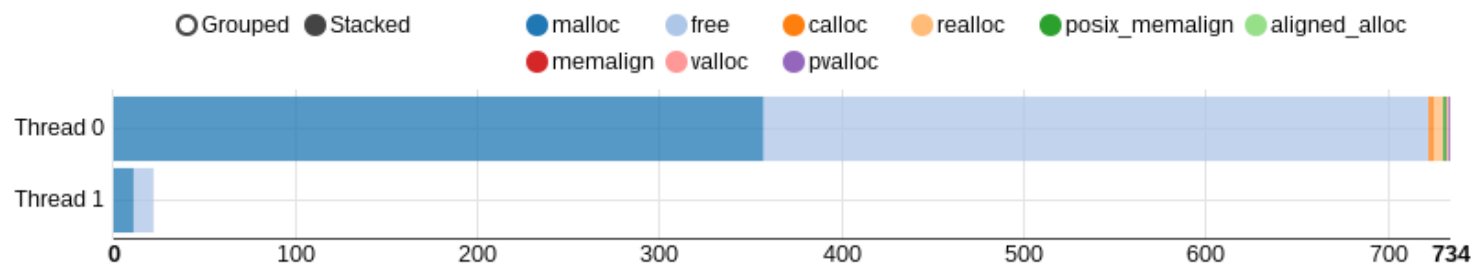
## Global statistics

Show all details Show help

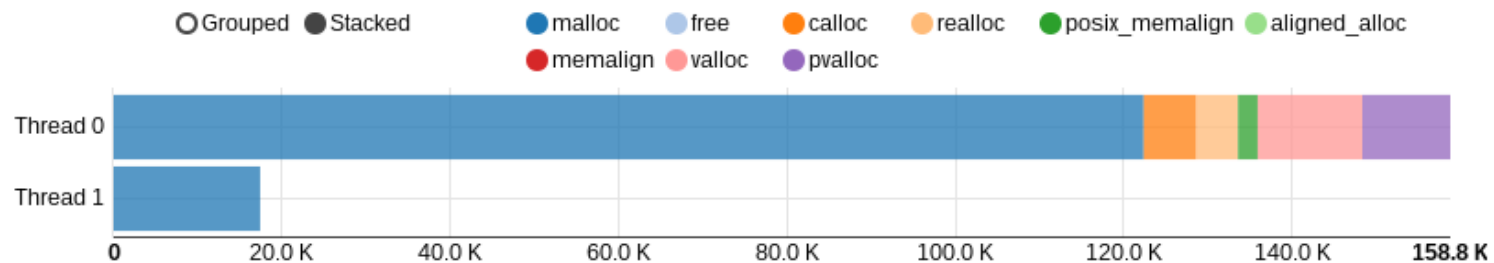
Physical memory peak	13.8 MB
Virtual memory peak	189.4 MB
Requested memory peak	12.1 MB
<b>Cumulated memory allocations</b>	<b>144.0 GB</b> ⚠
Allocation count	49.2 K
<b>Recycling ratio</b>	<b>12204.8</b> ⚠
Leaked memory	6.1 MB
Largest stack	96 B
Global variables	43.8 KB
TLS variables	40 B
<b>Global variable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

# Per thread statistics

## Call per thread



## Time per thread



# Source annotation

The screenshot displays the MALT WebView interface. On the left, a sidebar shows a list of symbols with their corresponding line numbers and allocated counts. The main area shows the source code of a C++ file with various annotations. A call stack is visible at the bottom right, showing the sequence of function calls leading to the current line. A summary table at the bottom left provides memory-related statistics.

Inclusive/Exclusive

Metric selector

Per line annotation

Call stacks reaching the selected site.

Symbols

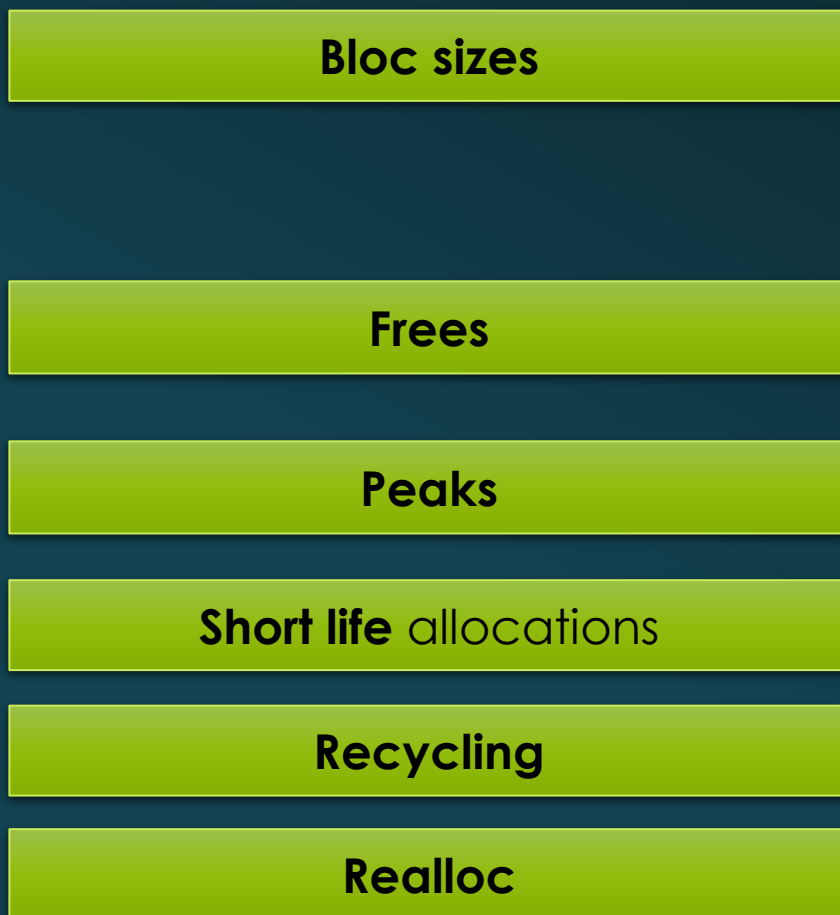
Details of symbol or line

Inclusive	
Allocated memory	6.2 KB
Freed memory	6.2 KB
Local peak	1.1 KB
Leaks	0
99 alloc	[ 64 B , 64 B , 64 B ]
99 free	[ 64 B , 64 B , 64 B ]
Lifetime	[ 18.5 K , 1.0 M , 12.3 M ] (cycles)

Function	Metric
_start	99
__libc_start_main_impl	99
__libc_start_call_main	99
main	99
recurseA(int)	99
recurseA(int)	99
malloc	11
recurseA(int)	88
malloc	11
recurseA(int)	77
malloc	11

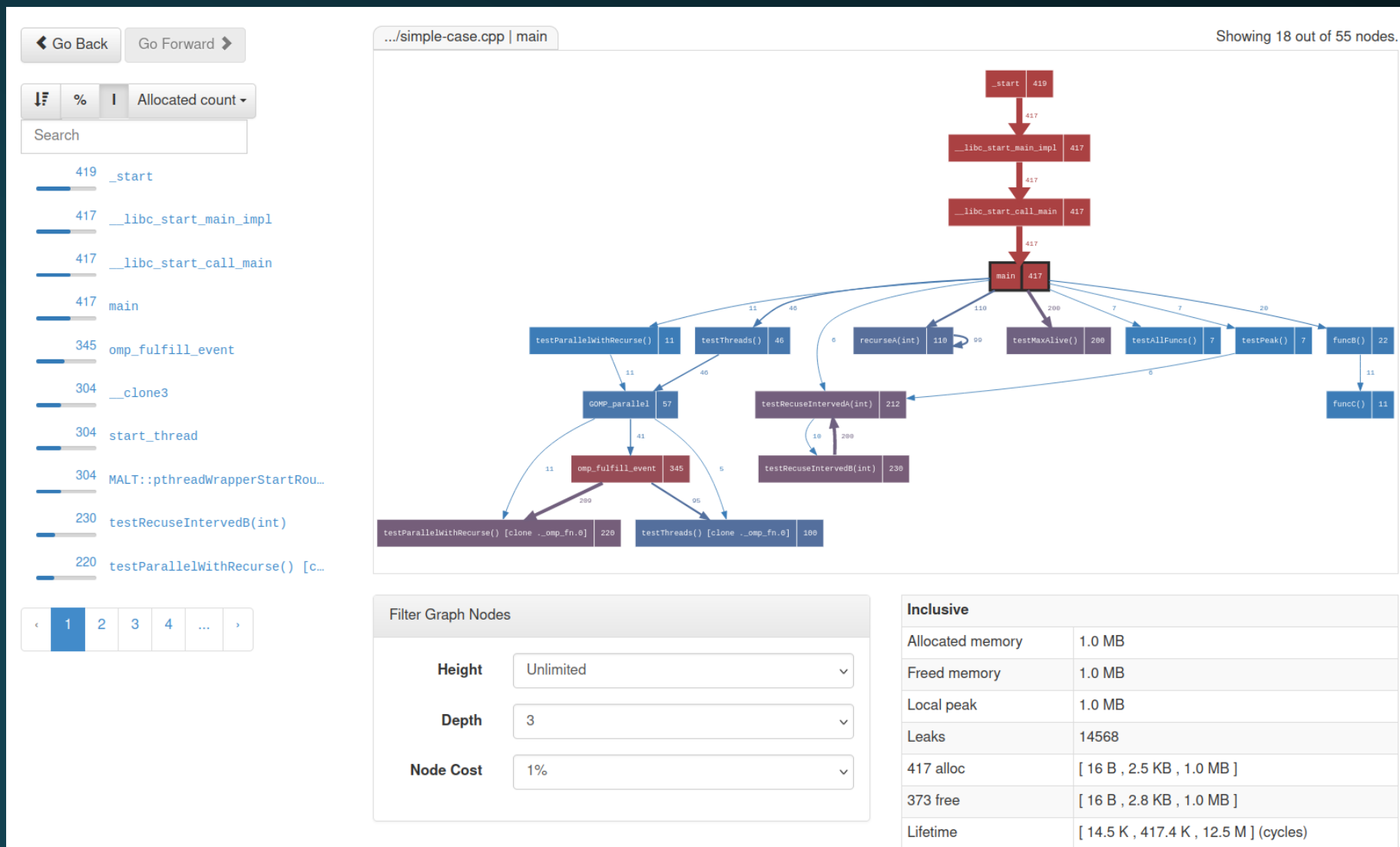


# Metrics



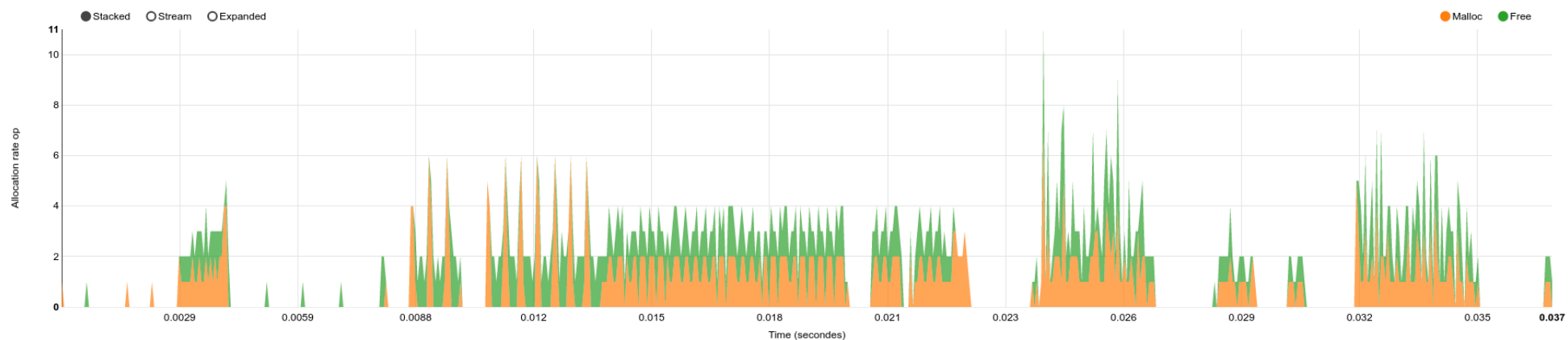
The screenshot shows a memory analysis tool interface. At the top, there are icons for sorting (downward arrow), percentage (%), and a filter icon (vertical bar). To the right, a dropdown menu is open, displaying a list of metrics. The metrics listed are: Allocated mem., Allocated count, Min. alloc size, Mean alloc size, Max. alloc size, Freed mem., Free count, Memory ops., Local peak, Global peak, Leaks, Max lifetime, Min lifetime, Recycling ratio, Realloc count, and Realloc sum. The background of the screenshot shows a table with columns for memory usage and function names, with some entries like `rt_main_impl` and `rt_call_main` visible.

# Call stack tree



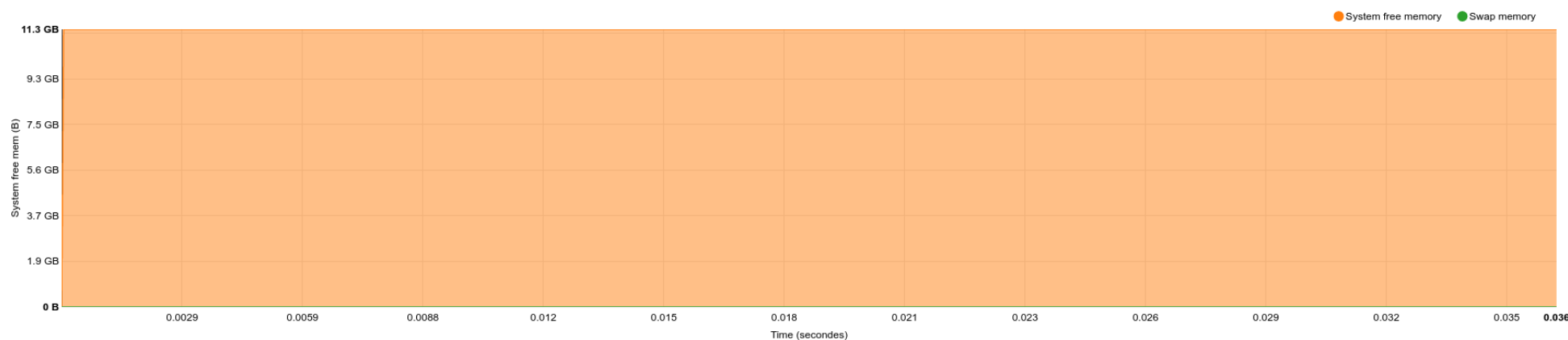
# Time charts

## Allocation rate (operations)



[JSON data](#) | [CSV data](#) | [Save as SVG](#)

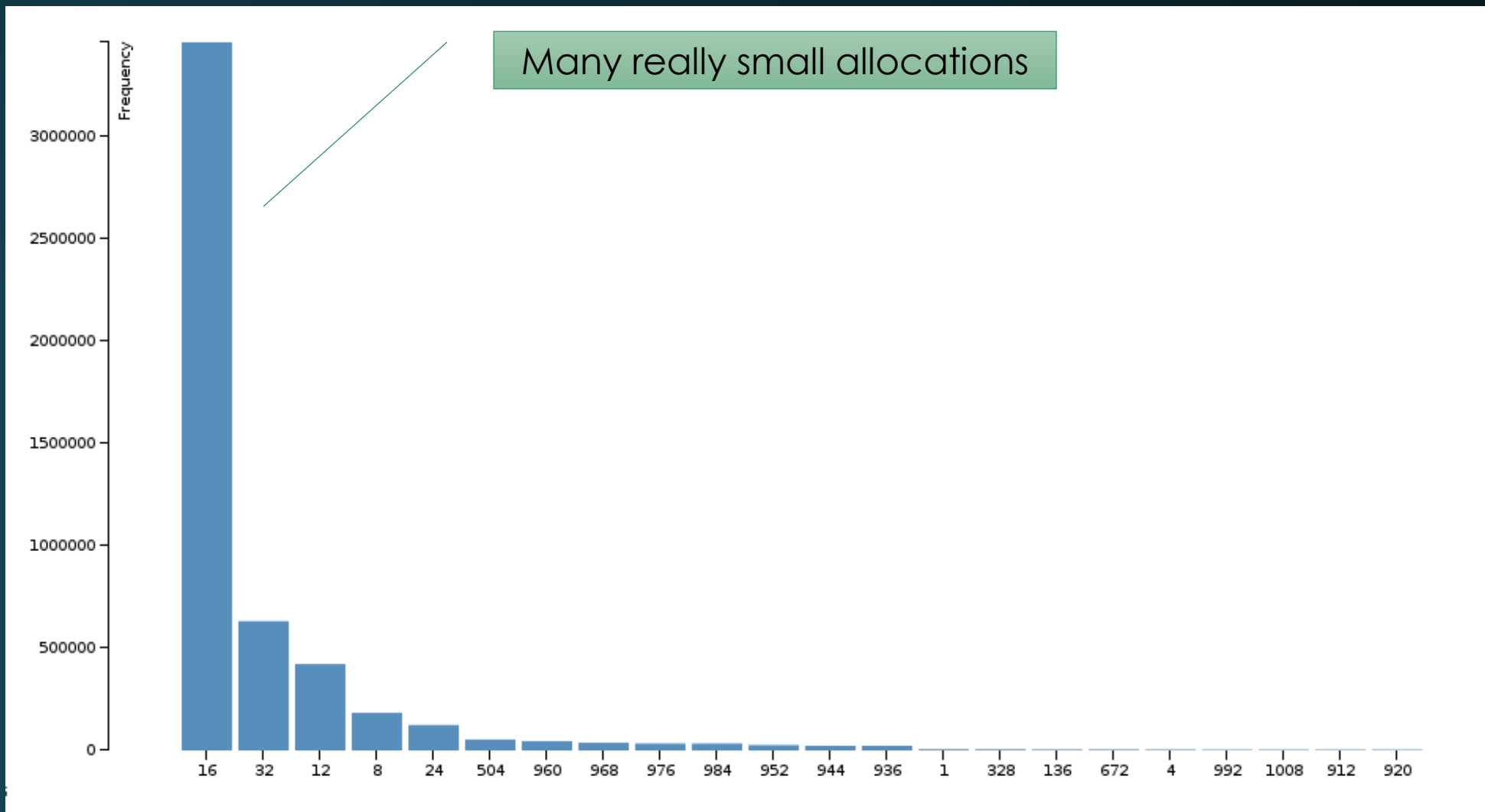
## System memory



[JSON data](#) | [CSV data](#) | [Save as SVG](#)

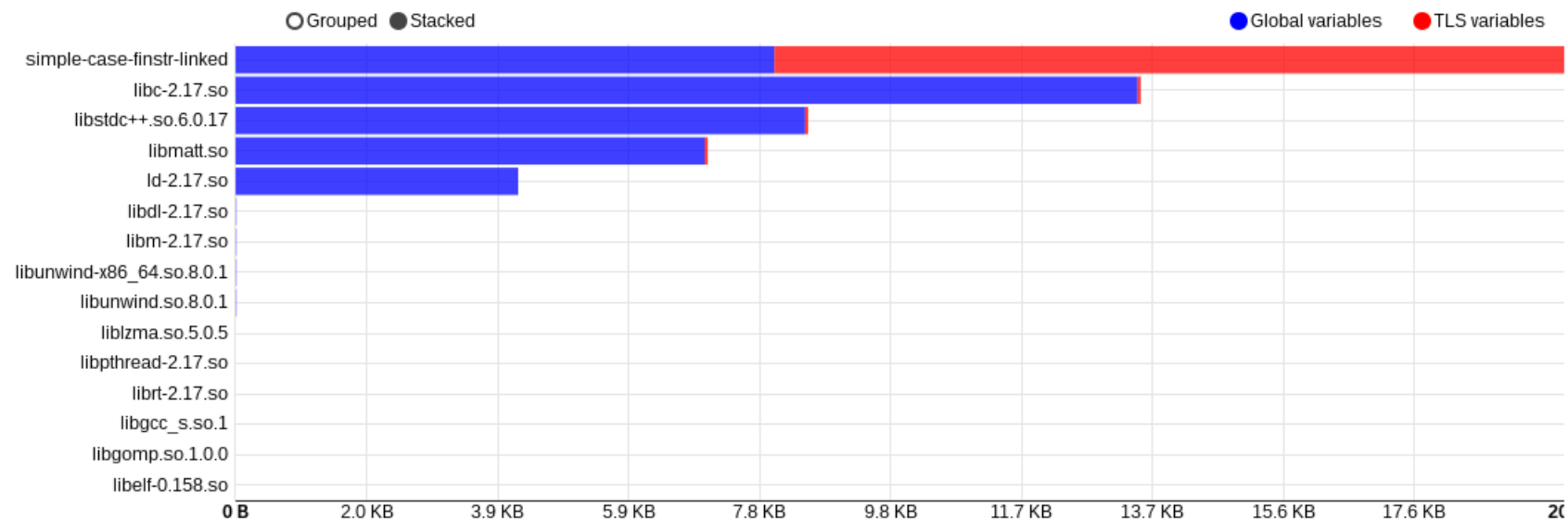
# Chunk size distribution

## Example from YALES2 with gfortran issue

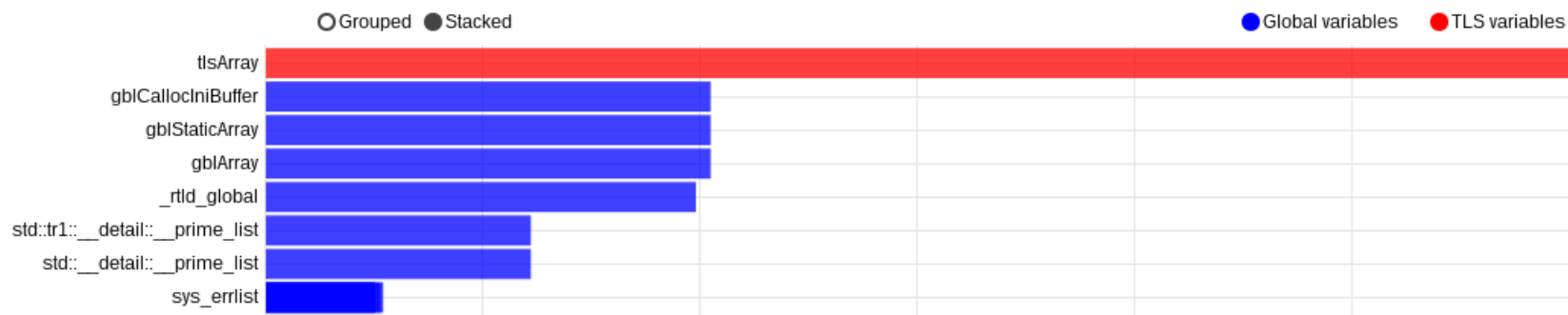


# Global variables

## Distribution over binaries



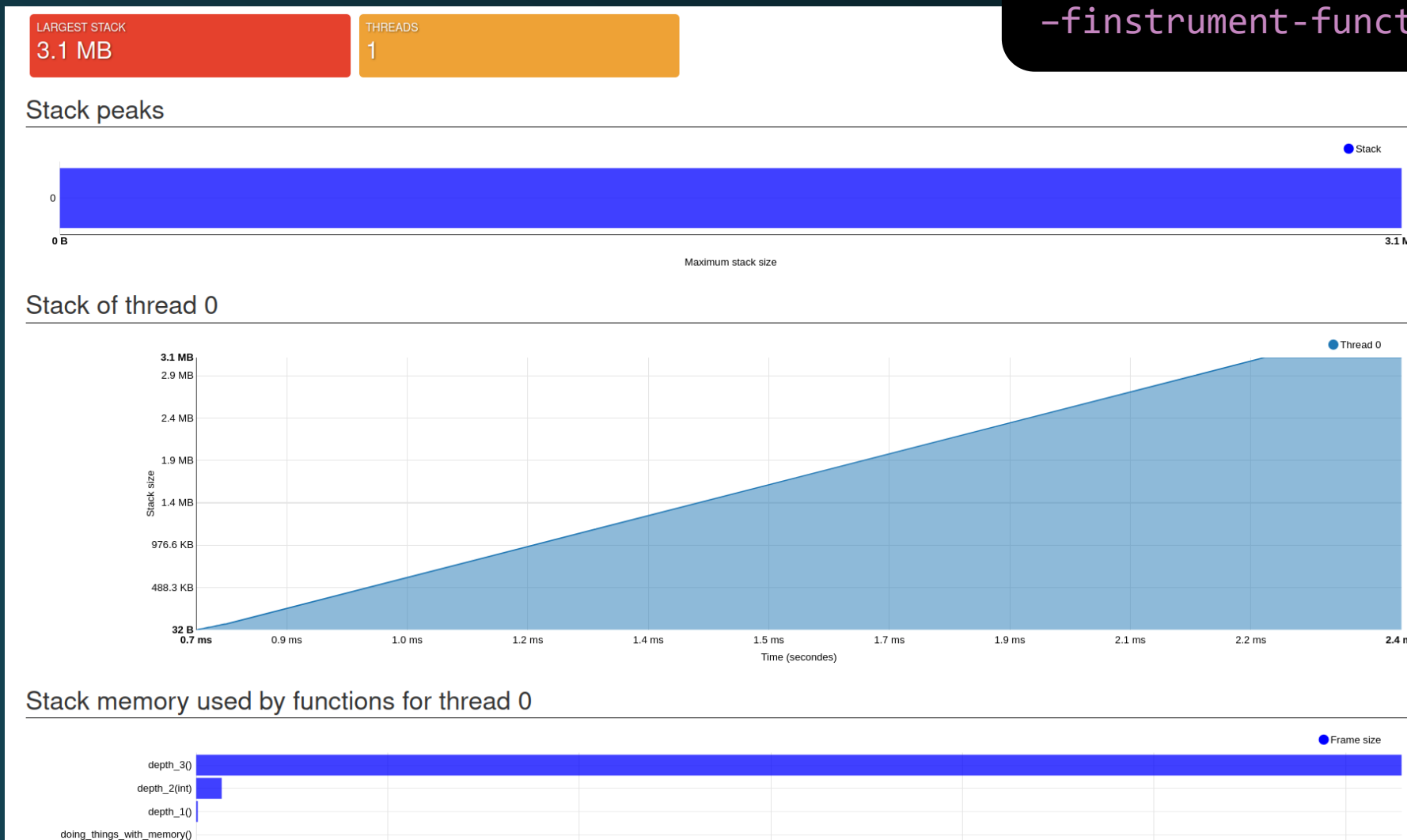
## Distribution over variables





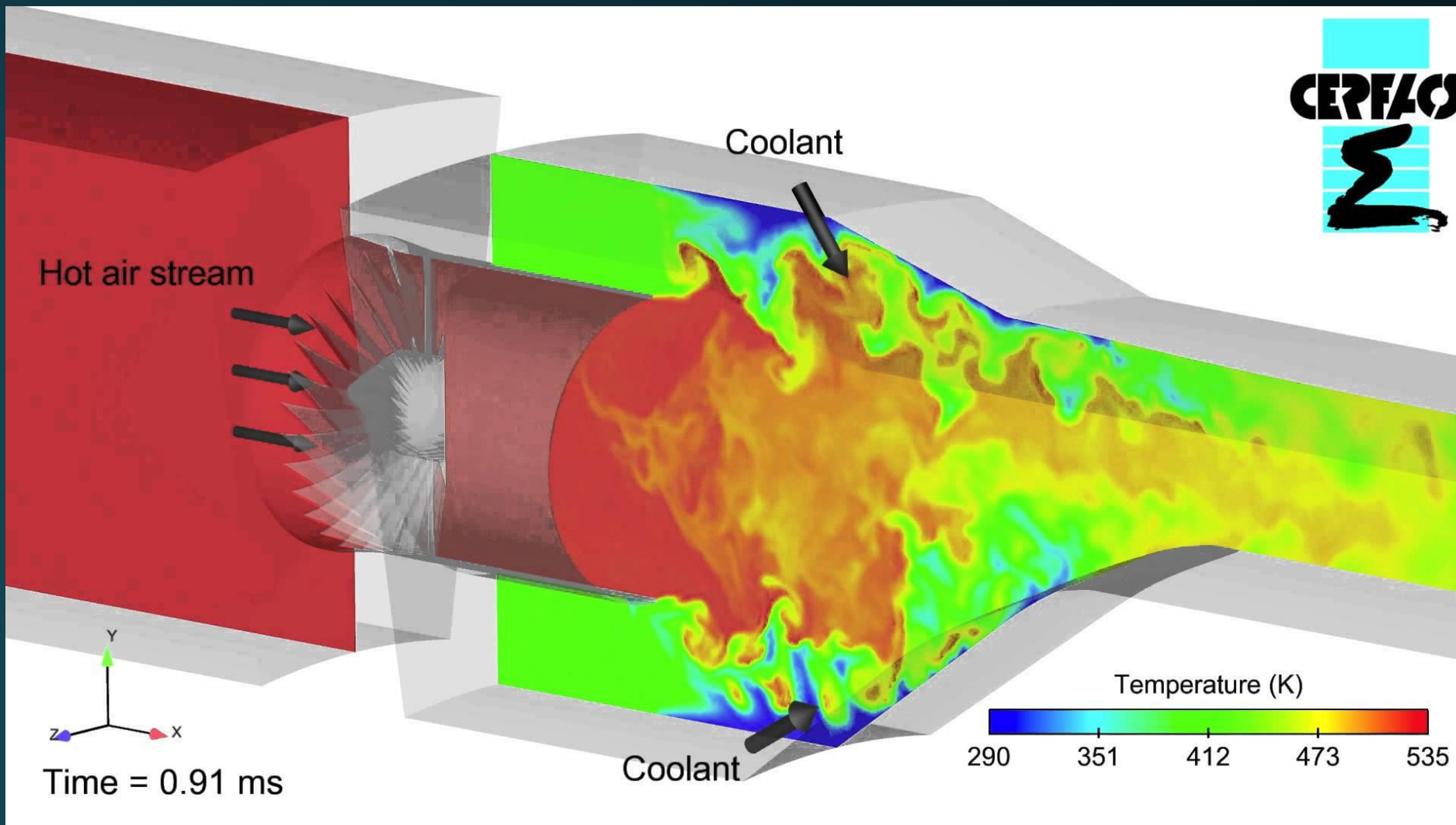
# Memory used by stacks

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



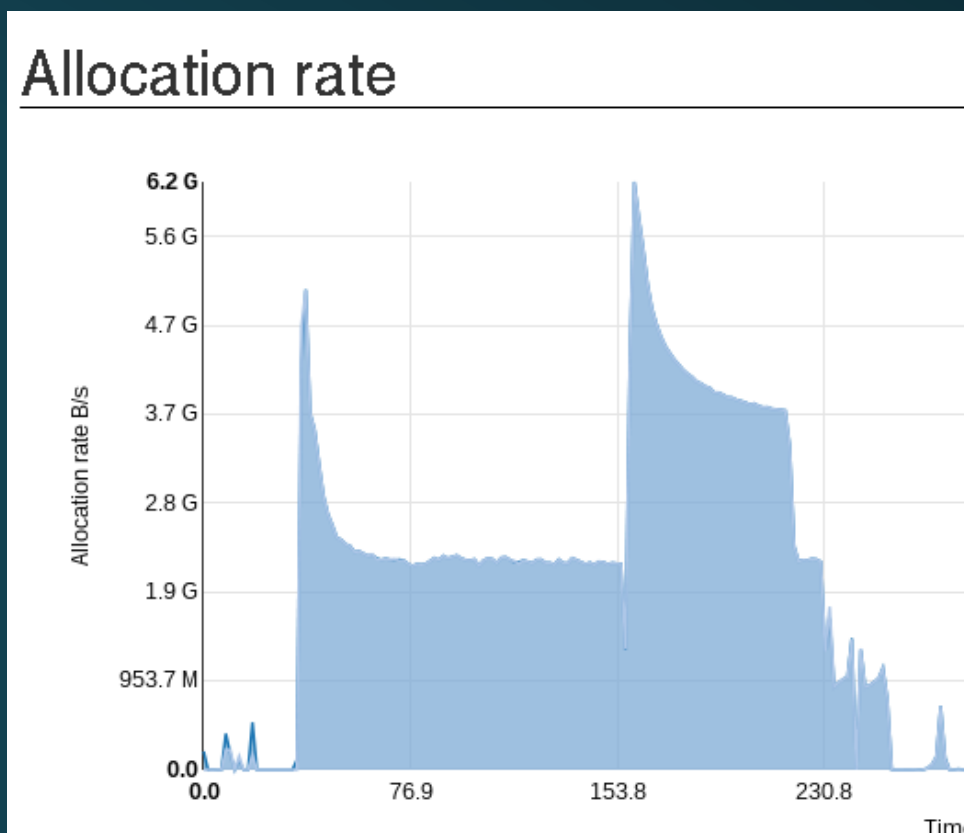
# Real cases

# Cerfacs - AVBP- CFD simulator



# Example on AVBP init phase

- Issue with reallocation on init



```

99      CALL assert(capacity==size(array),&
100                  'array and capacity variable are not
101                  '
102      IF (needed_size>capacity) THEN
103          IF (ALLOCATED ( temp) ) DEALLOCATE(temp)
104          ALLOCATE ( temp(capacity))
105
106          DO i=1,capacity
107              temp(i)=array(i)
108          END DO
109
110          DEALLOCATE ( array)
111          ALLOCATE ( array(new_cap))
112
113          DO i=1,capacity
114              array(i)=temp(i)
115          END DO
116
117          capacity=new_cap
118      END IF
119 <
  
```

57 GB

57 GB

**Total :**

Allocated memory : 56.8 GB  
 Max alive memory : 135.7 M  
 3.5 K alloc : [ 16.0 KB , 16.3 MB , 33.7 MB ]  
 Lifetime : [ 107.8 K , 26.7 M , 476.7 M ] (cycles)

**Own :**

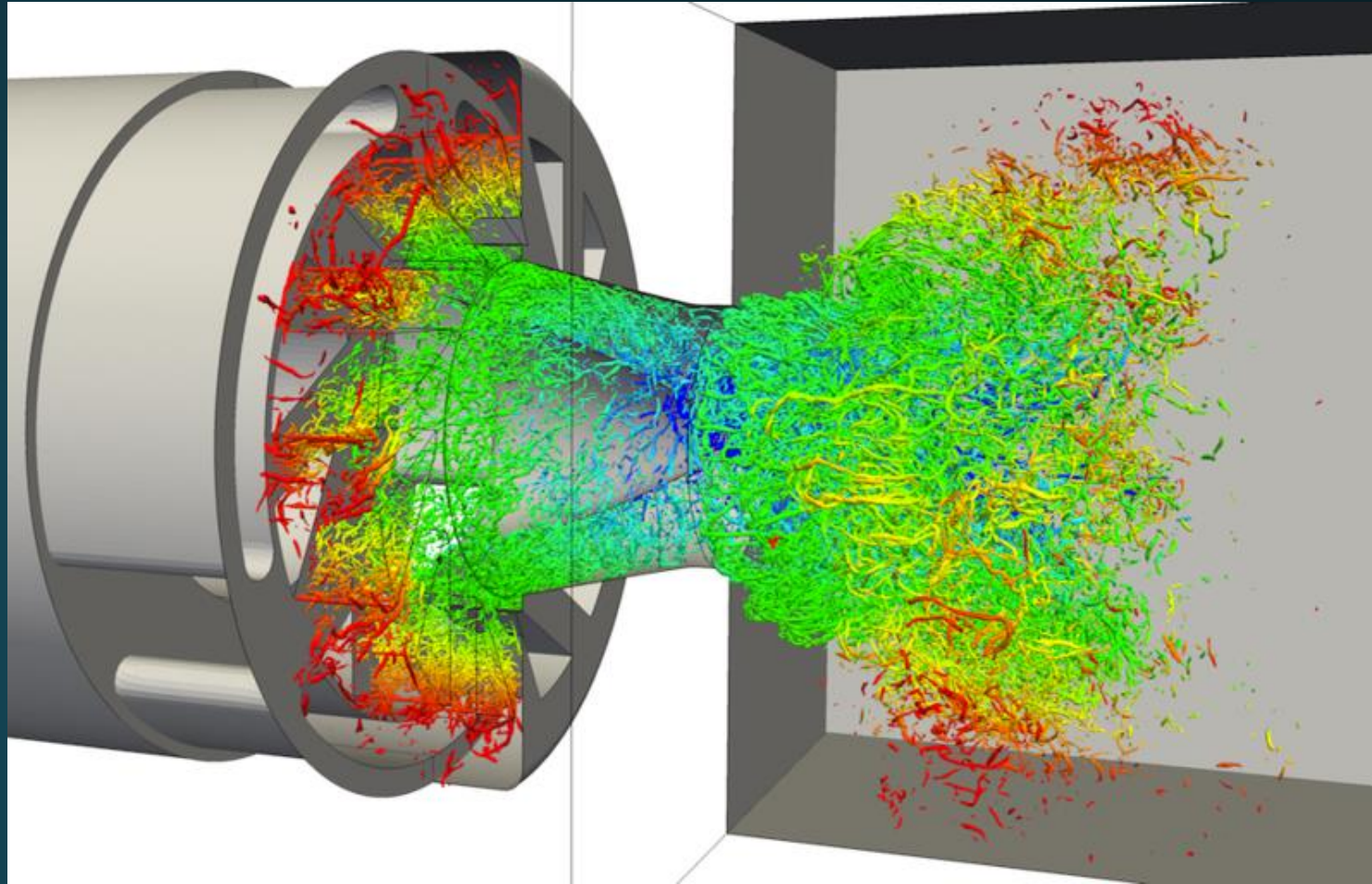
Allocated memory : 56.8 GB  
 Max alive memory : 135.7 M  
 3.5 K alloc : [ 16.0 KB , 16.3 MB , 33.7 MB ]  
 Lifetime : [ 107.8 K , 26.7 M , 476.7 M ] (cycles)

Function

▶ \_start



# Coria – YALES2 - Combustion





# Allocatable arrays on YALES2

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

MATT WebView

Allocation count ▾

Search

- 911.9 K data\_comm\_m::copy\_i...
- 896.4 K data\_comm\_m::copy\_...

Search intensive alloc functions

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

```

892      do i=1,nitem_el_grp
893         el_grp_ind = el_grp_index2int_comm_index%val(1,i)
894         int_comm_ind = el_grp_index2int_comm_index%val(2,i)
608 K 895         el_grp_r2%val(1:dim1,el_grp_ind) = int_comm_r2%val(1:dim1,int_comm_ind)
896      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 !

MATT WebView

%  I  ←  → Min. size ▾

Search

- 1.0 B /usr/lib/gcc/x86\_64-p...
- 1.0 B \_\_strdup
- 1.0 B data\_defs\_m::resize\_...

Search for the minimal chunk size.

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

```

530
531
532
533
534
1 B 535
536
537
538
539

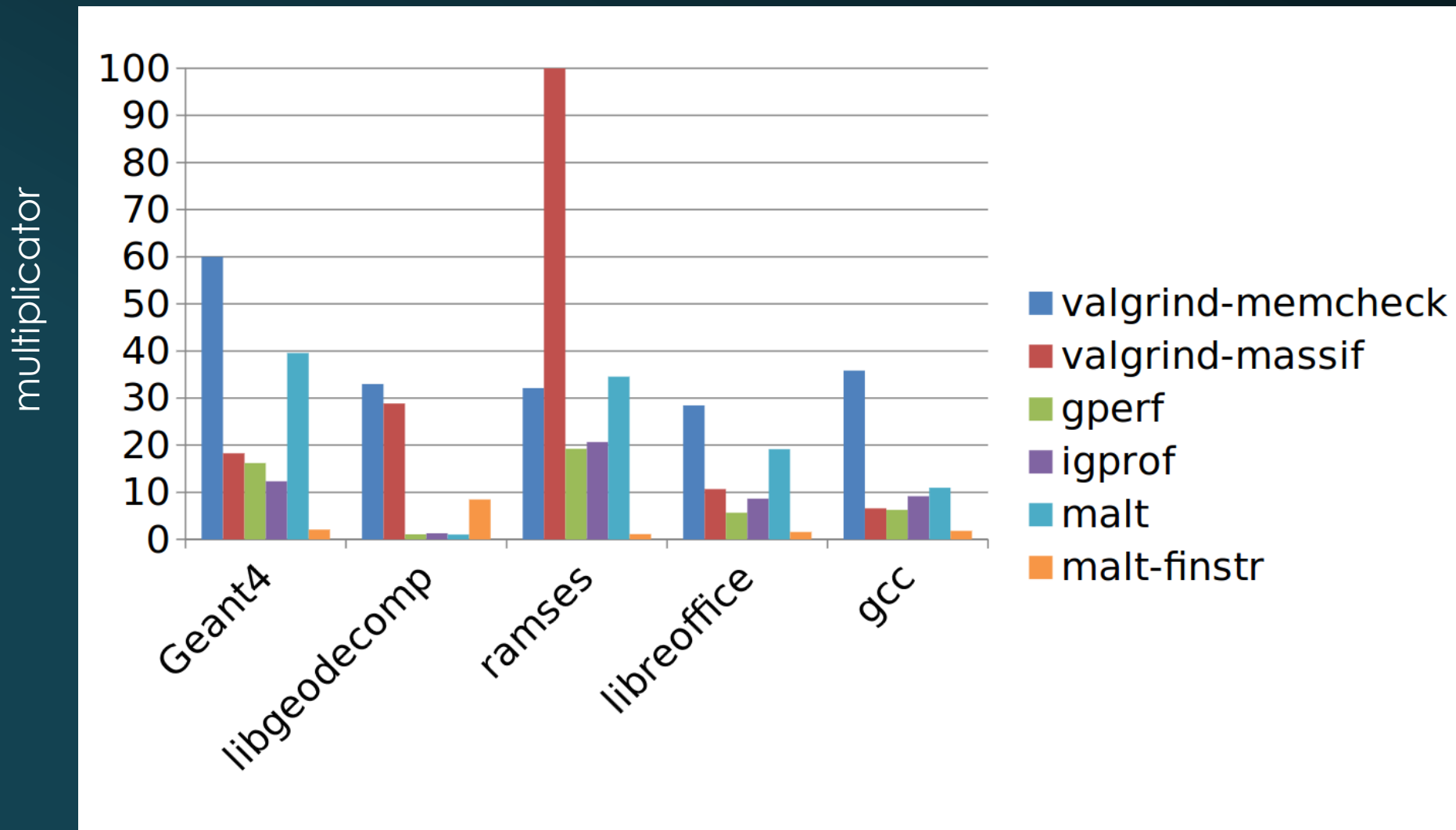
```

```

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
allocate(data_ptr%r2_ptrs(nel_grps))
do n=1, nel_grps
    NULLIFY(data_ptr%r2_ptrs(n)%ptr)
end do

```

# Overhead

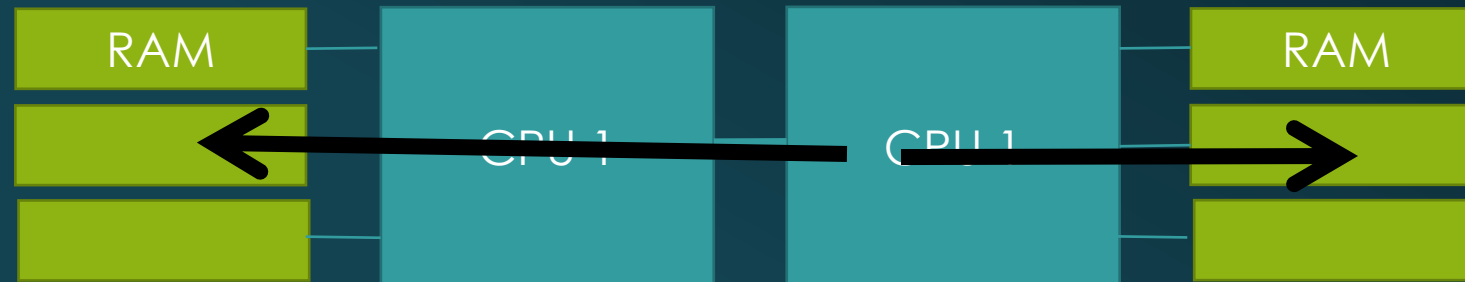


# Reminder on NUMA

Non Uniform Memory Access

# What is NUMA ?

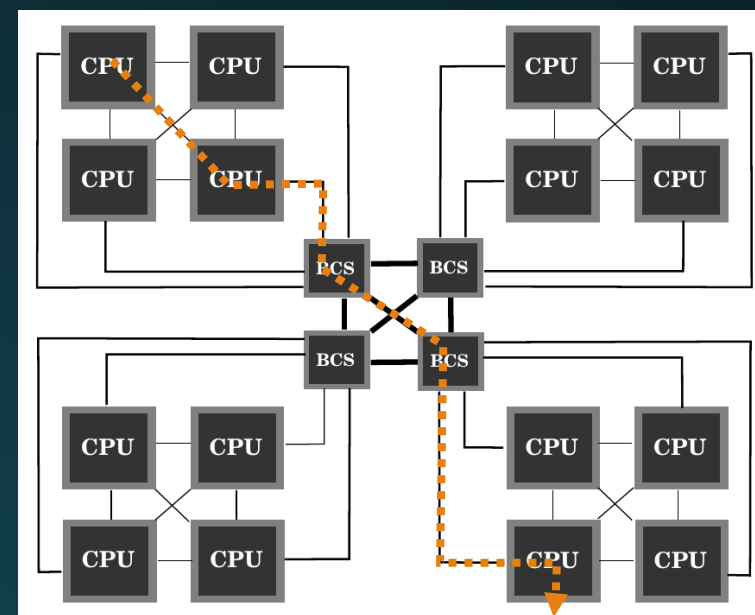
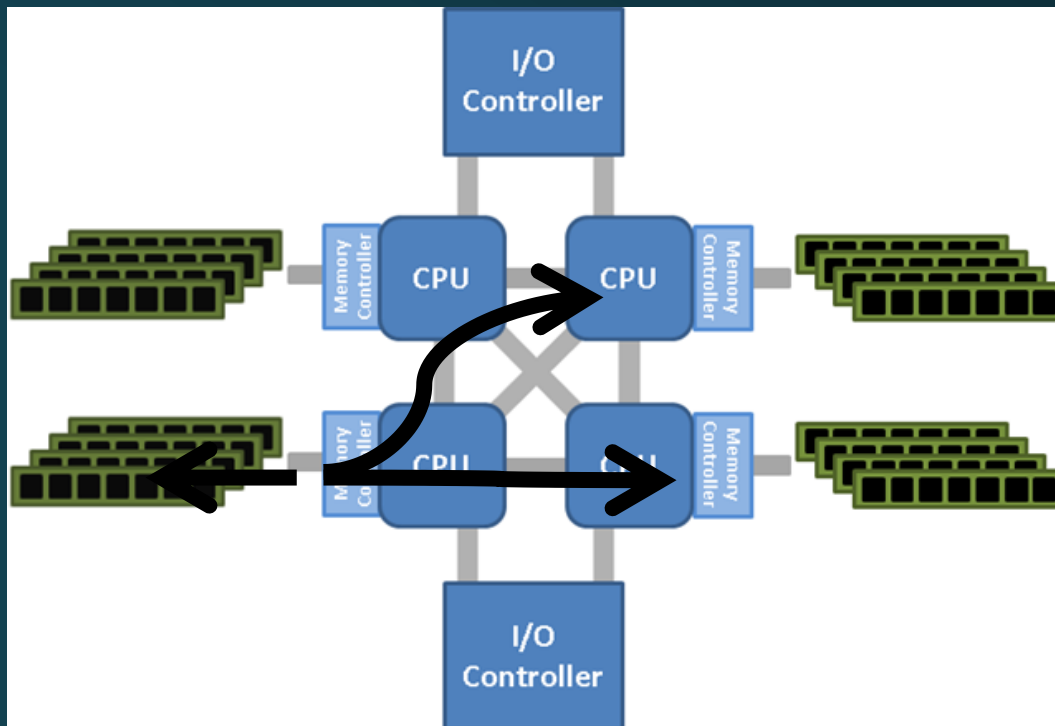
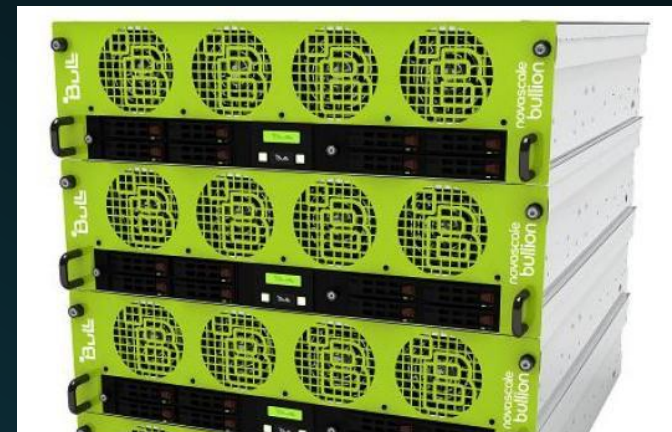
- Each CPU has its **own memory**
- Access to **remote memory** we need to **go through the owner CPU**





# Architecture

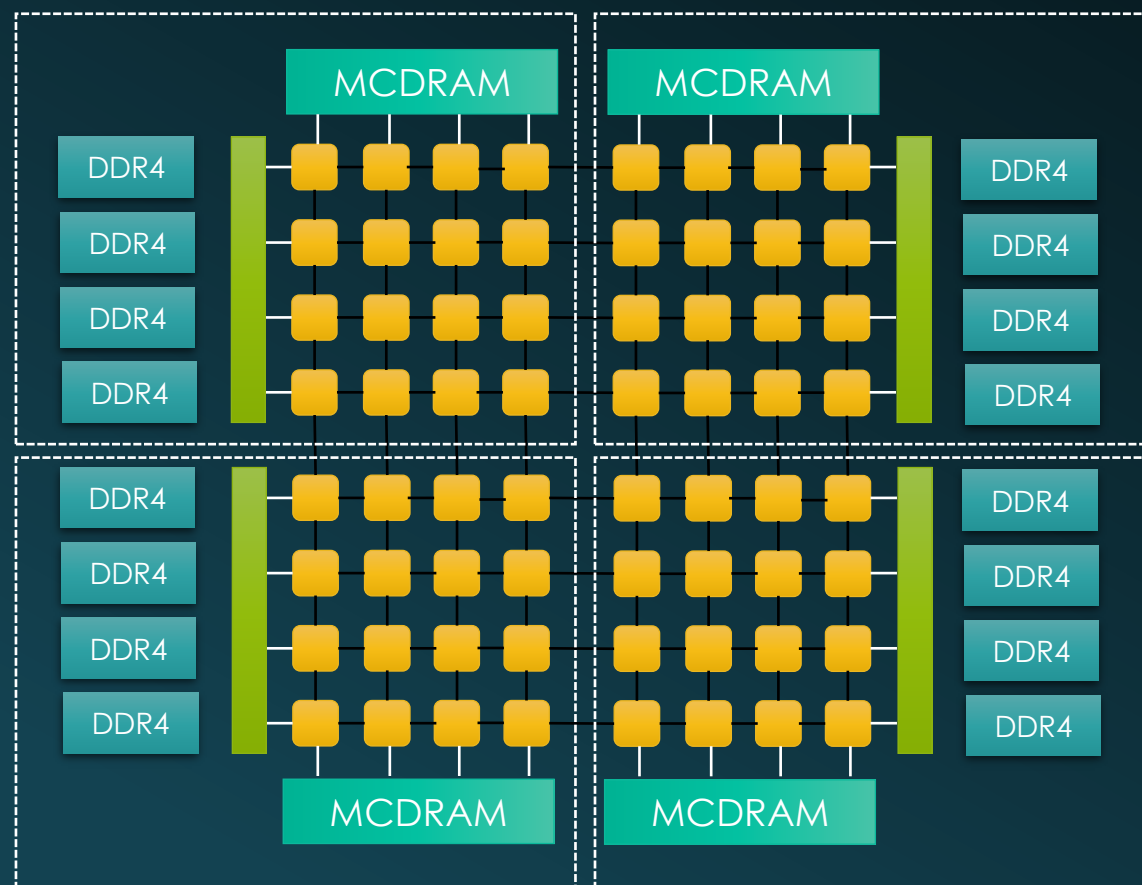
➤ A bit more complex



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

# Today topology

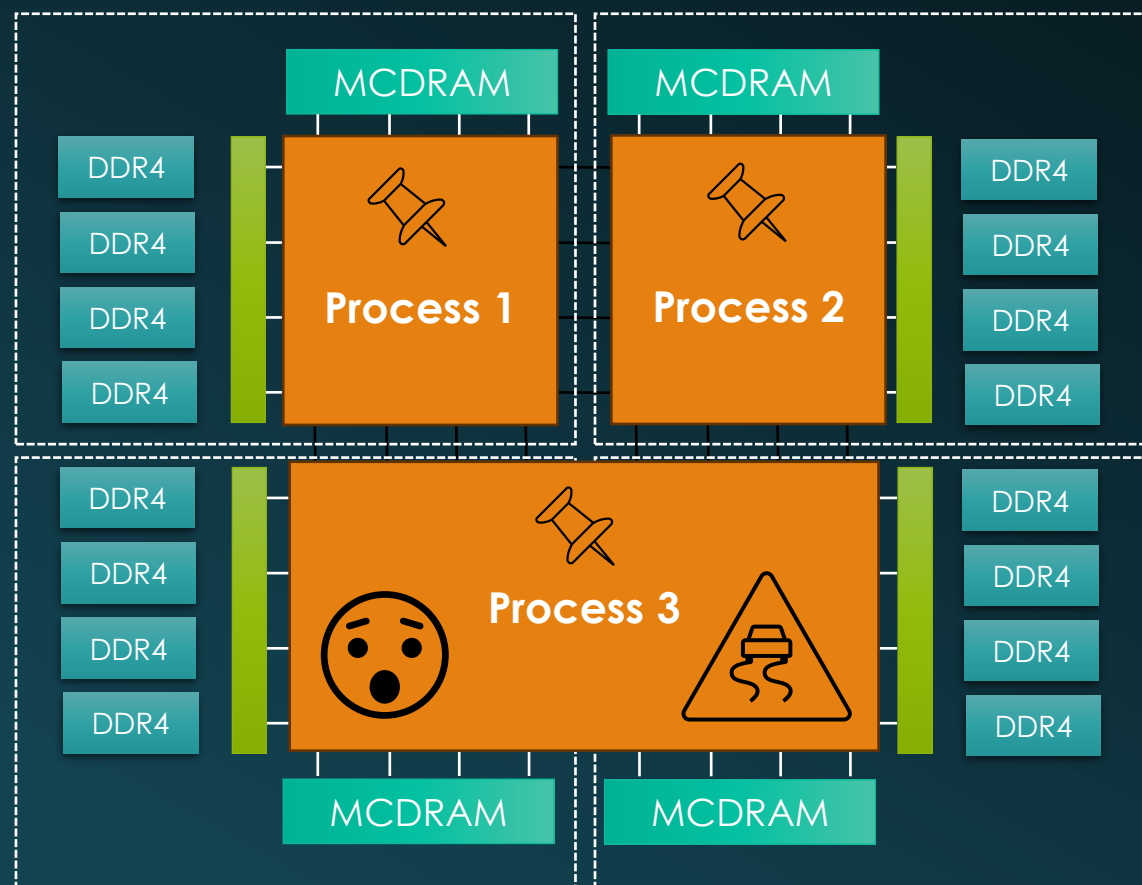
- Intel Knight Landing, mode **SNC2** or **SNC4**
- Also add fast memory **MCDRAM (HBM)** presented as **NUMA** or **LLC cache**



# Today topology

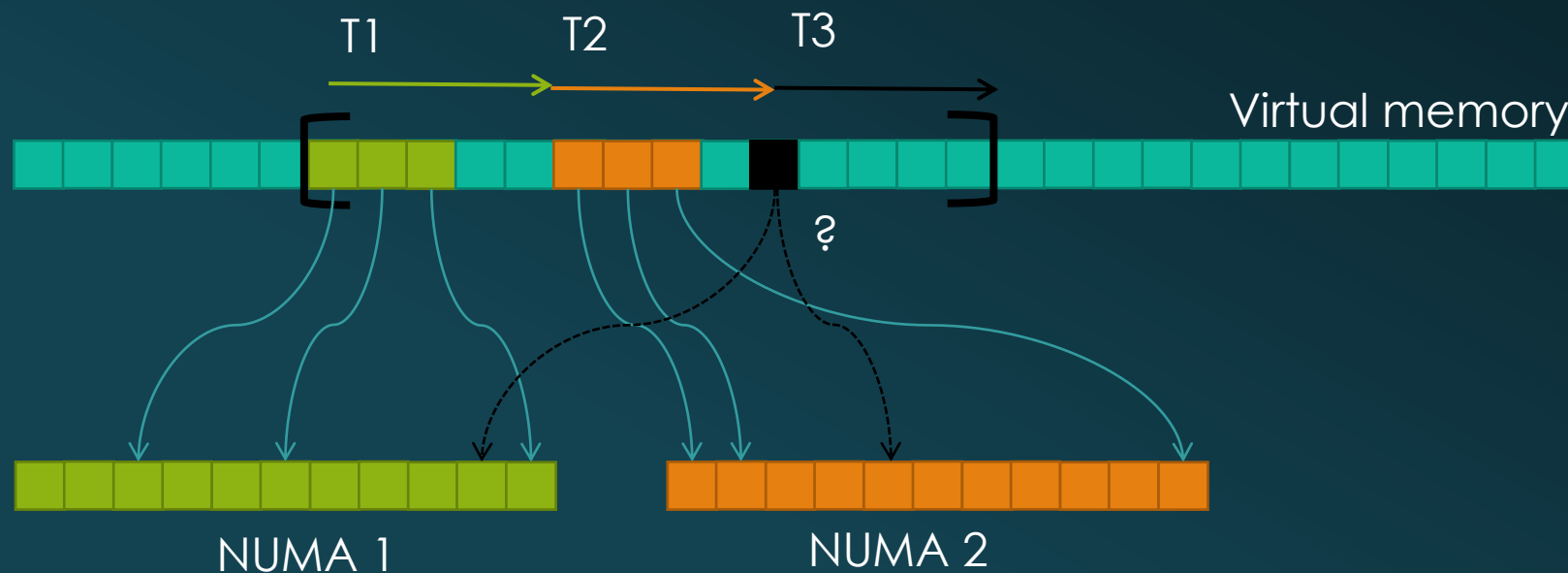
Safe usage :

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

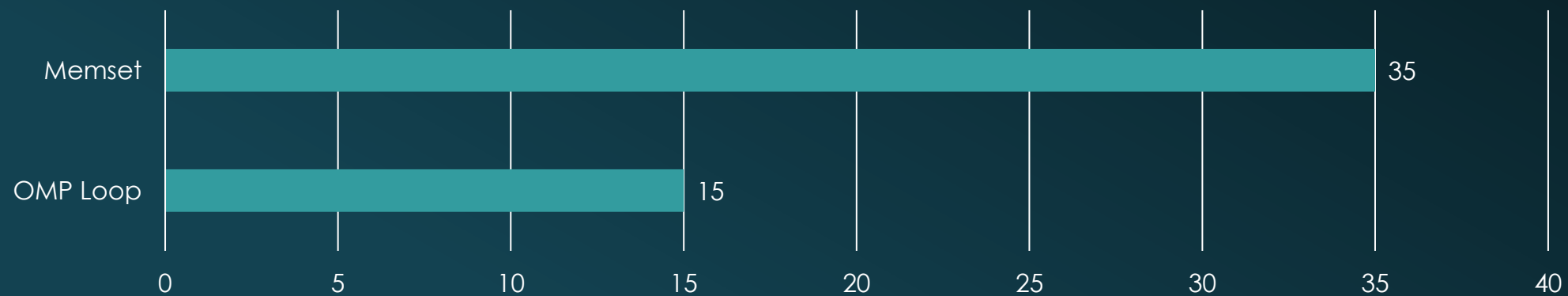
```
#pragma omp parallel for  
for (int i = 0 ; i < SIZE ; i++)  
    array[i]++;
```

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



# Performance impact

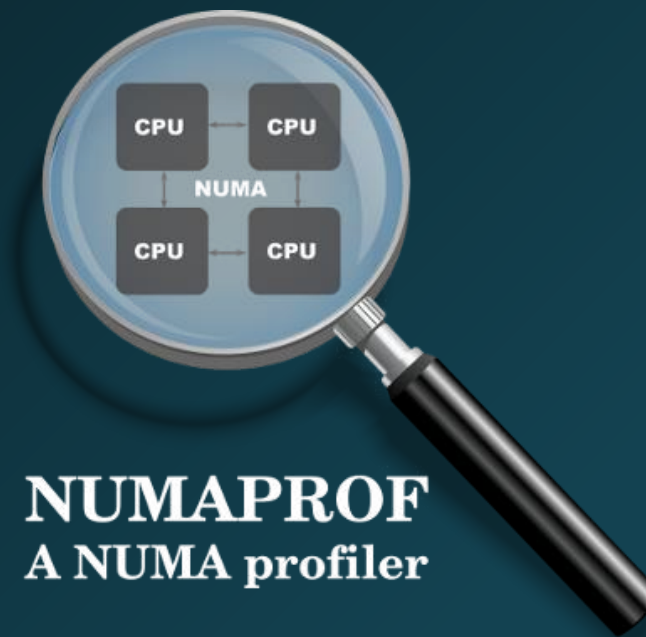
➤ On a real machine (2 NUMA) :



# 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**





**NUMAPROF**  
A NUMA profiler

# NumaProf

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

# 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.....

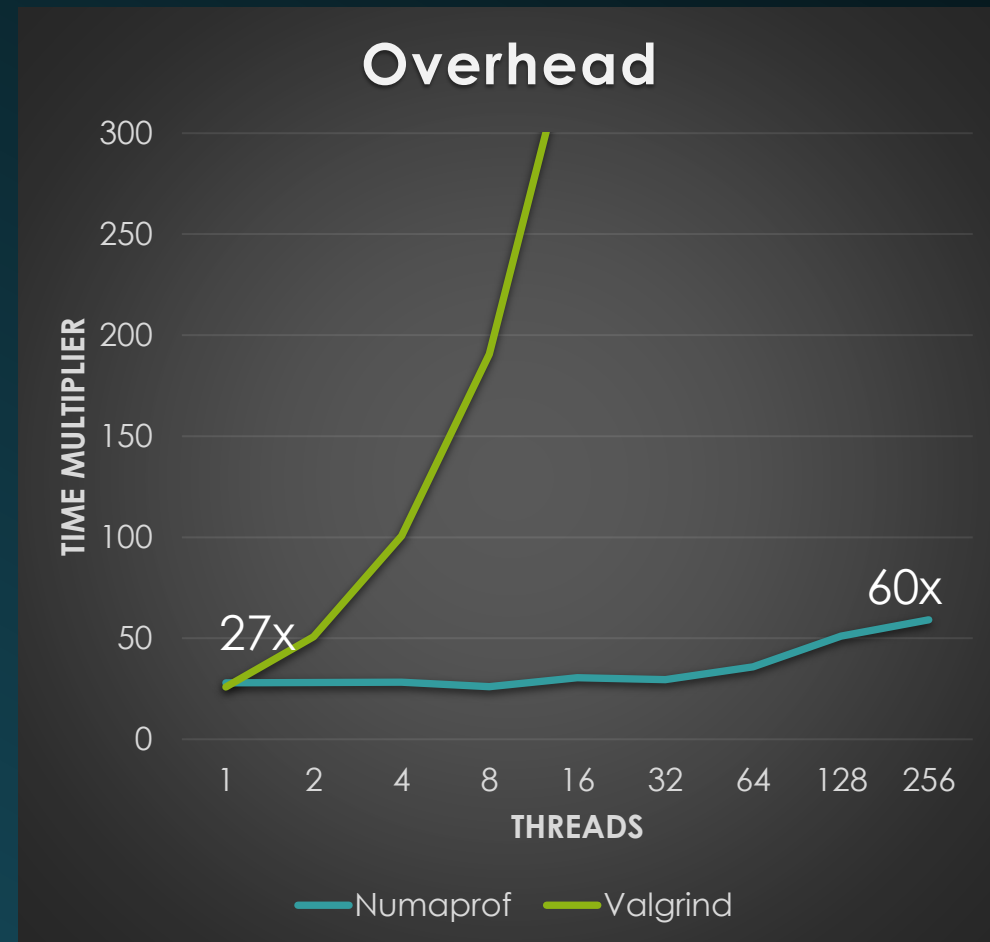
# On access we need...

- 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

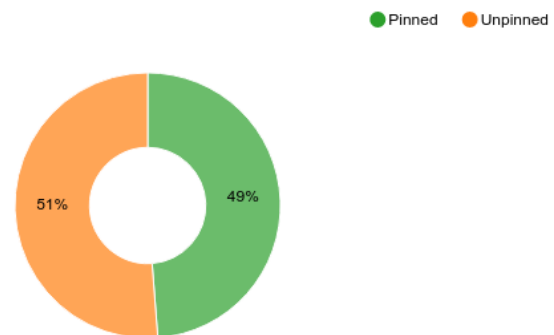
- Of course overhead is large: **~30x**
- But is **scale**
- Example code hydro on **KNL**:



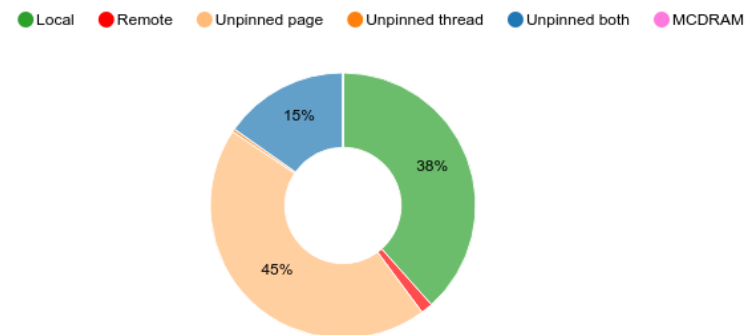
# GUI and example

# Global summary

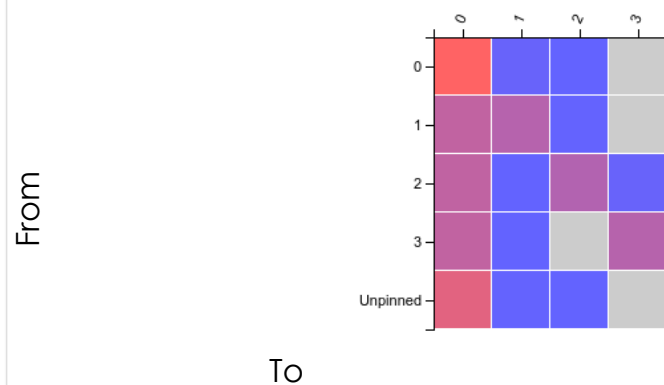
First touch



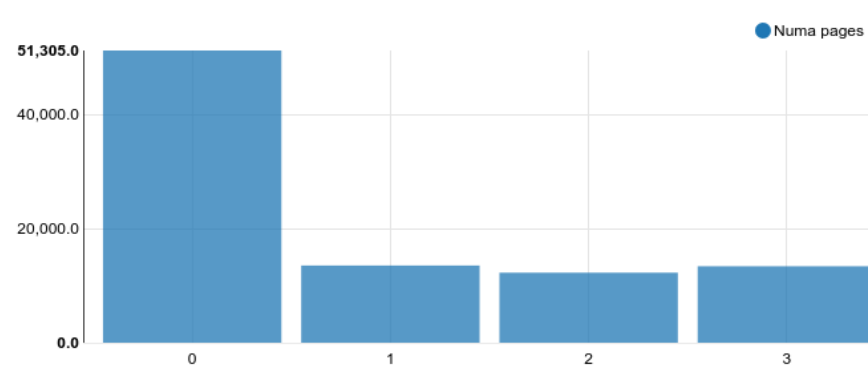
Memory access



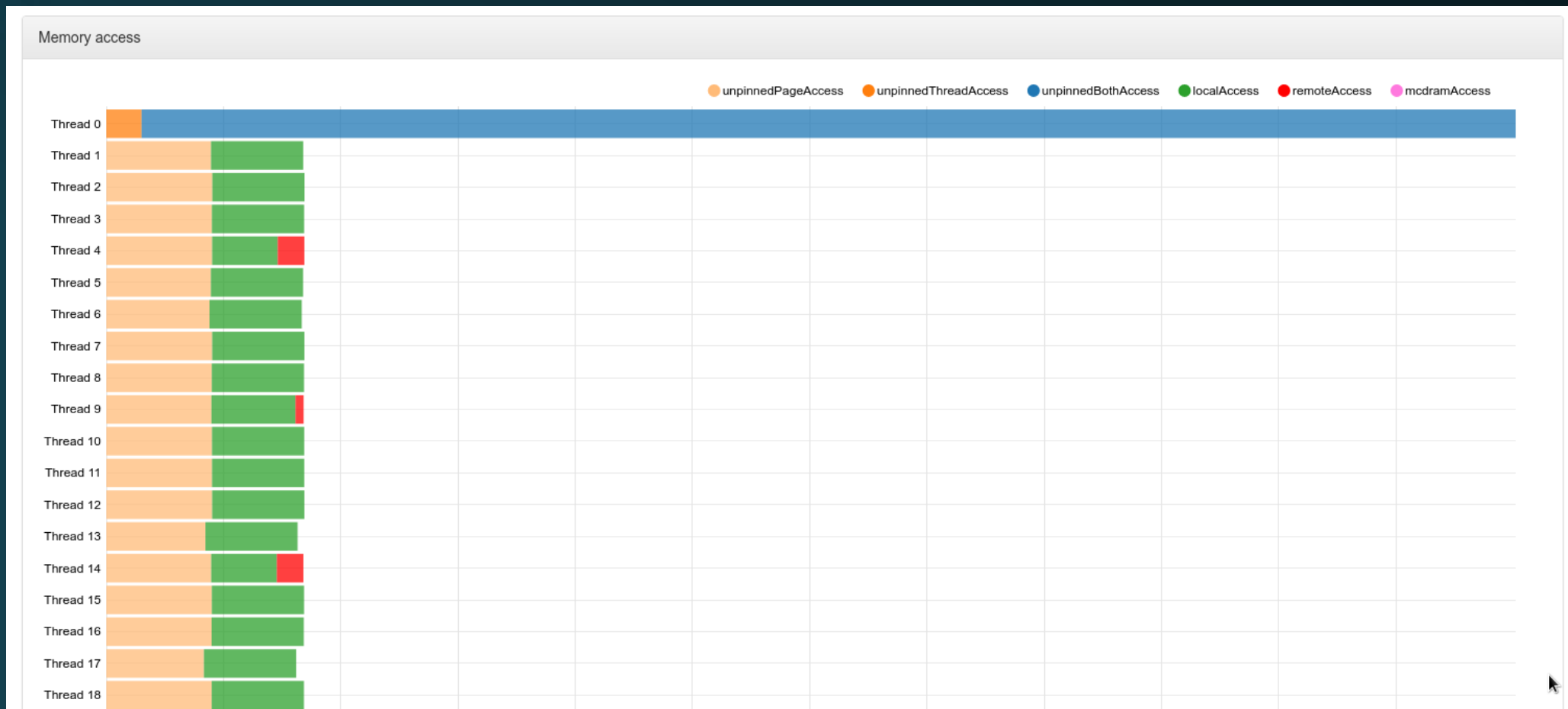
Access matrix



Peak allocated numa pages

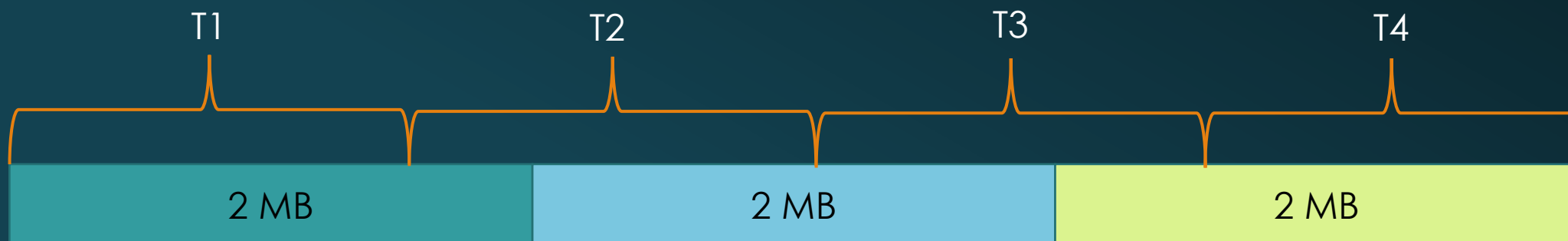


# Statistics per thread



# OMP and huge pages

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




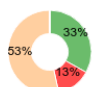
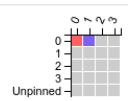




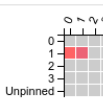
# Details per thread

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	<p>At 64.31%, pin thread on node 0</p> <p>At 64.31%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND</p>

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	<p>At 64.32%, pin thread on node 1</p> <p>At 64.32%, do memory binding MPOL_DEFAULT on -1 considered as NO_BIND</p>

# Source & asm annotations

Numaprof Home Threads Details Sources Assembler Help

% All access ▾

Search

104.9 M	badFirstAccess(unsigned long) [cl...	80
104.9 M	betterFirstAccess(unsigned long) ...	104.9 M
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	40
77.8 K	strcmp	52.4 M
69.9 K	_dl_relocate_object	80
42.4 K	check_match.9440	104.9 M

```

23         buffer[i] = 0;
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;

```

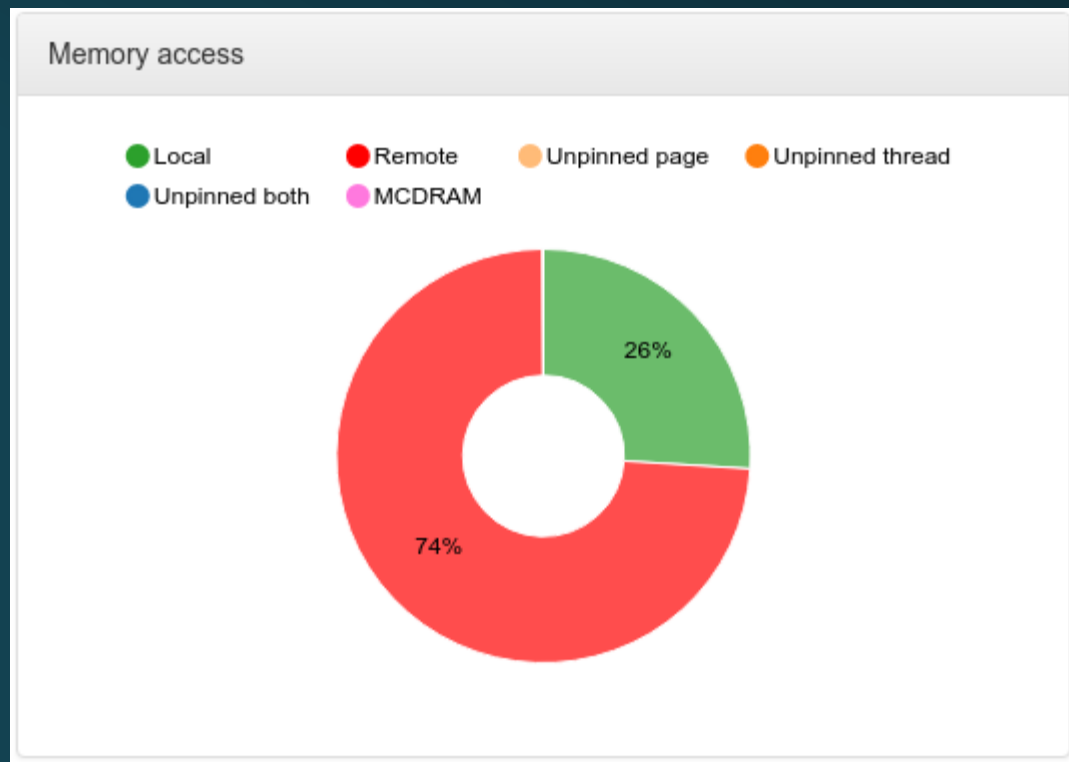
Line 41

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

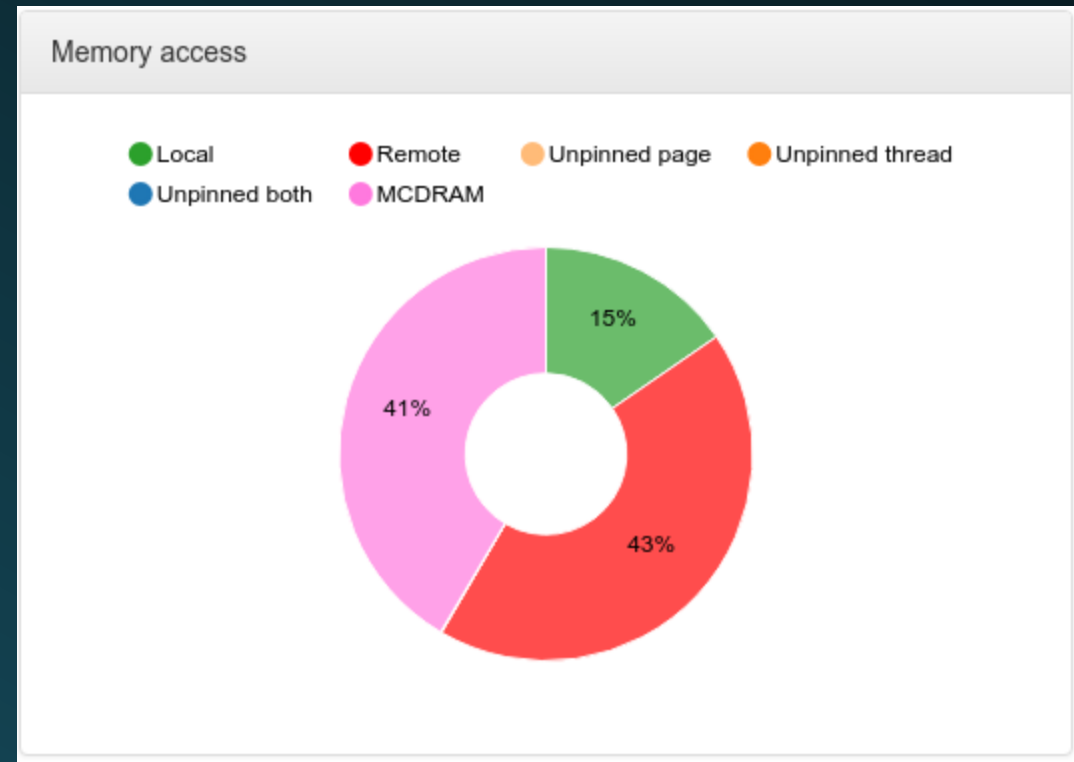
52 int main()

# Code Hydro

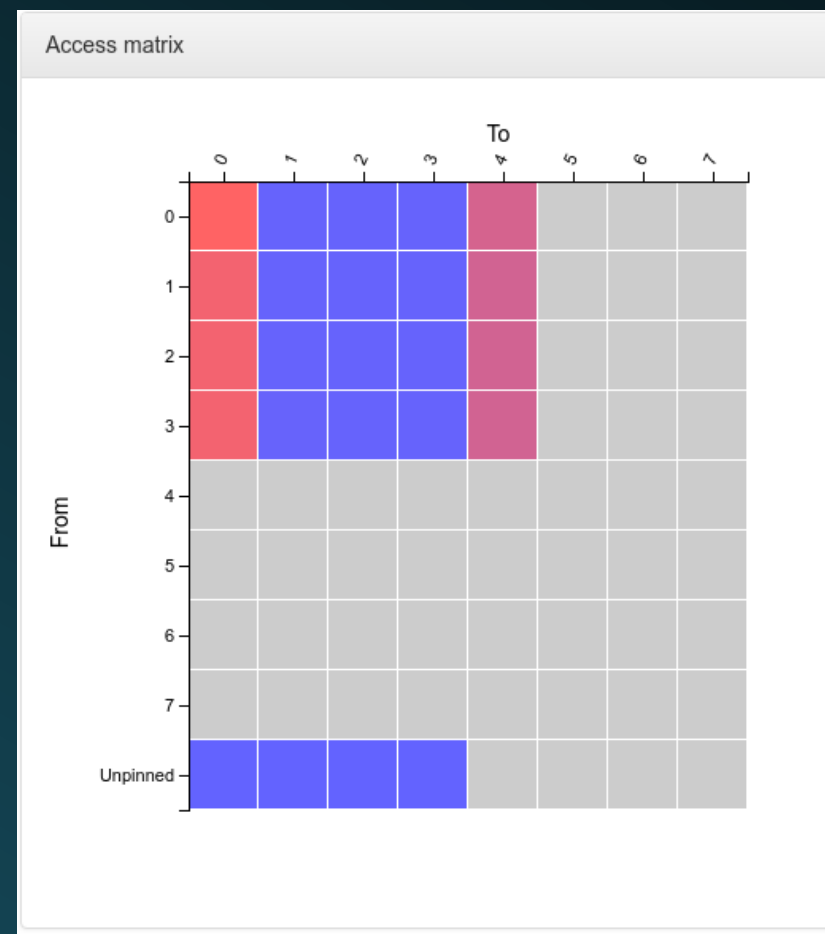
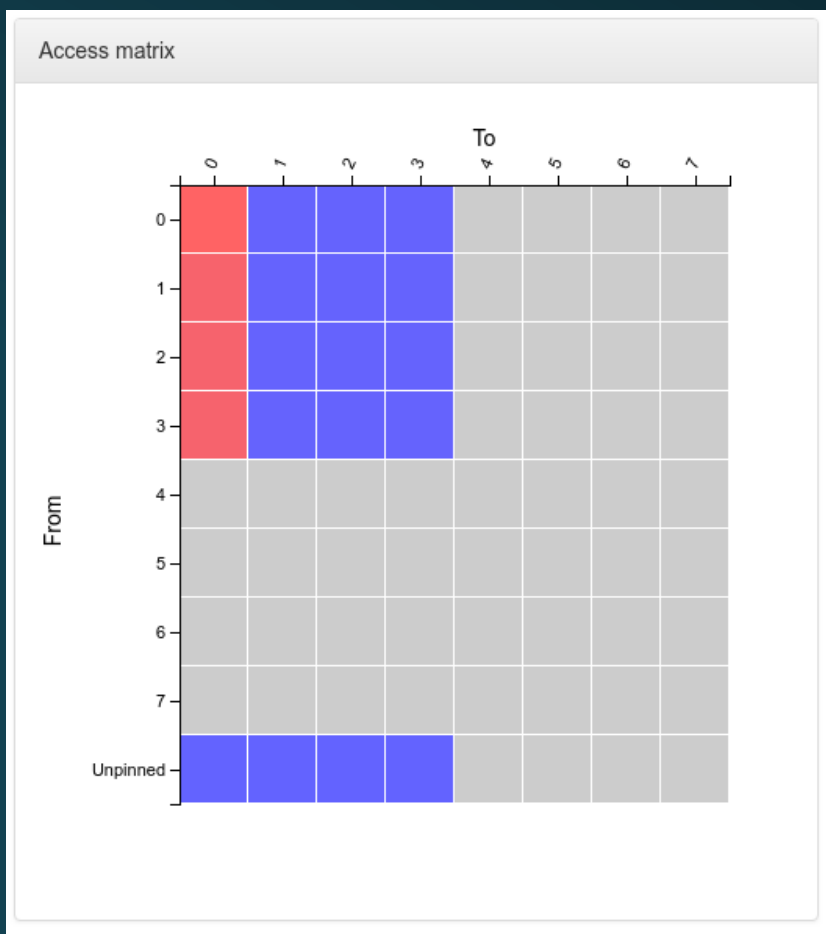
➤ KNL Without HBM



WITH HBM



# Original Hydro access matrix



# Ordering issue

% Alloc remote access ▾

Search

11.9 G	gomp_malloc
8.4 G	Domain::setTiles() [clone ._omp_fn.0]
1.3 G	Soa::Soa(int, int, int)
2.6 M	Tile::initTile(Soa*)
1.9 M	main

```

/data/svalat/Projects/Hydro/HydroC/HydroCplusMPI/Domain.cpp | Domain::setTiles() [clone .
699
700     m_localDt = AlignedAllocReal(m_nbtiles);
701     m_tiles = new Tile *[m_nbtiles];
702     #pragma omp parallel for private(i) if (m_numa) SCHEDULE
703     for (int32_t i = 0; i < m_nbtiles; i++) {
704         m_tiles[i] = new Tile;
705     }
706     // Create the Morton holder to wander around the tiles
707     m_morton = new Matrix2 < int32_t > (mortonW, mortonH);
708     // cerr << mortonW << " " << mortonH << endl;
709     m_mortonIdx = m_morton->listMortonIdx();
710     assert(m_mortonIdx != 0);
711

```

```

#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

% 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
21  using namespace Soa;
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
32      m_q = new Soa(NB_VAR, lgx, lgy);
33      m_qxm = new Soa(NB_VAR, lgx, lgy);
34      m_qxp = new Soa(NB_VAR, lgx, lgy);
35      m_dq = new Soa(NB_VAR, lgx, lgy);
36      m_qleft = new Soa(NB_VAR, lgx, lgy);
37      m_qright = new Soa(NB_VAR, lgx, lgy);
38      m_qgdnv = new Soa(NB_VAR, lgx, lgy);
39
40      m_c = new Matrix2 < real_t > (lgx, lgy);
41      m_e = new Matrix2 < real_t > (lgx, lgy);
42

```

# Parallel allocations

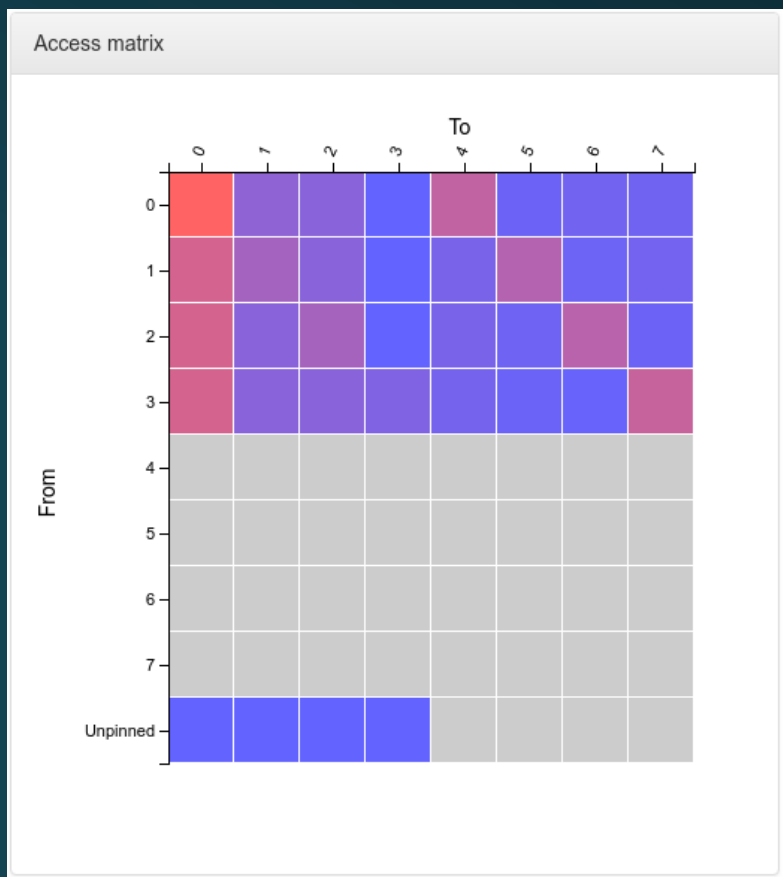
## ➤ Original

```
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



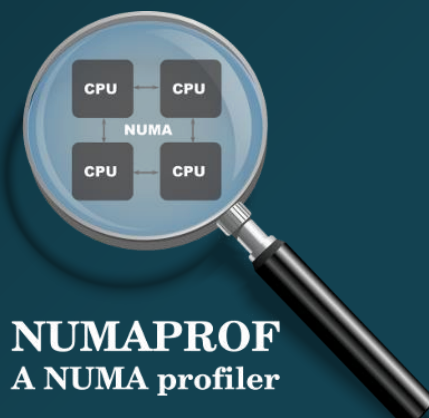
# Conclusion

- **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>

