

MALT & NUMAPROF, Memory Profiling for HPC Applications

SÉBASTIEN VALAT – FOSDEM 2019 – TRACK HPC

Origin of the tools

2

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



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



Motivation

3

- ▶ Lot of **issues** today :
 - ▶ **Huge** memory **space** to **manage** (~TB of memory)
 - ▶ **Lot more** distinct **allocations** (75 M in 5 minutes)
 - ▶ **Multi-threading** : 256 threads
 - ▶ **Hidden** into large (**huge**) C/C++/Fortran **codes** (~**1M** lines).
- ▶ Access:
 - ▶ **NUMA** (Non Uniform Memory Access)
 - ▶ **Memory wall** !

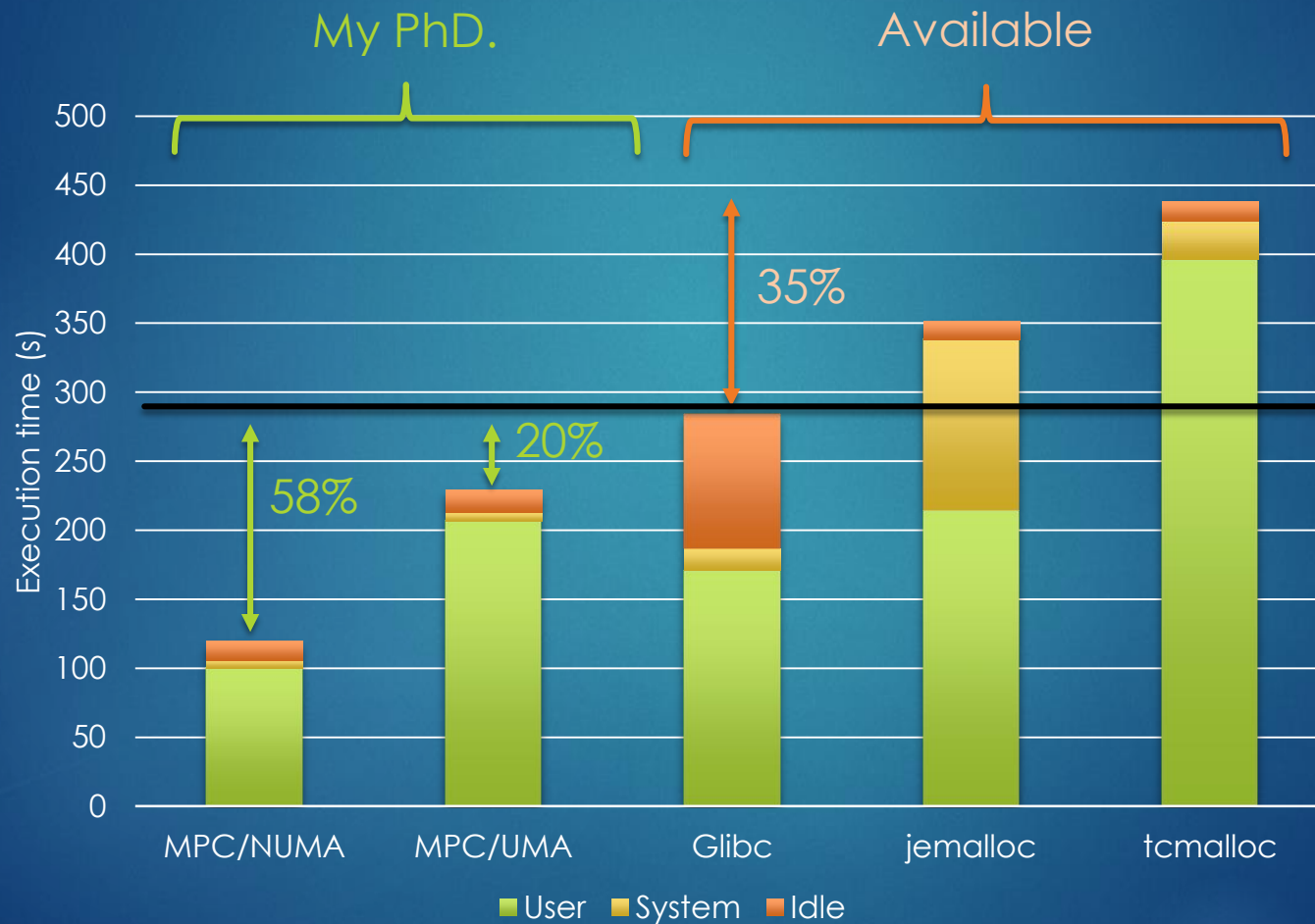
Key today

4

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

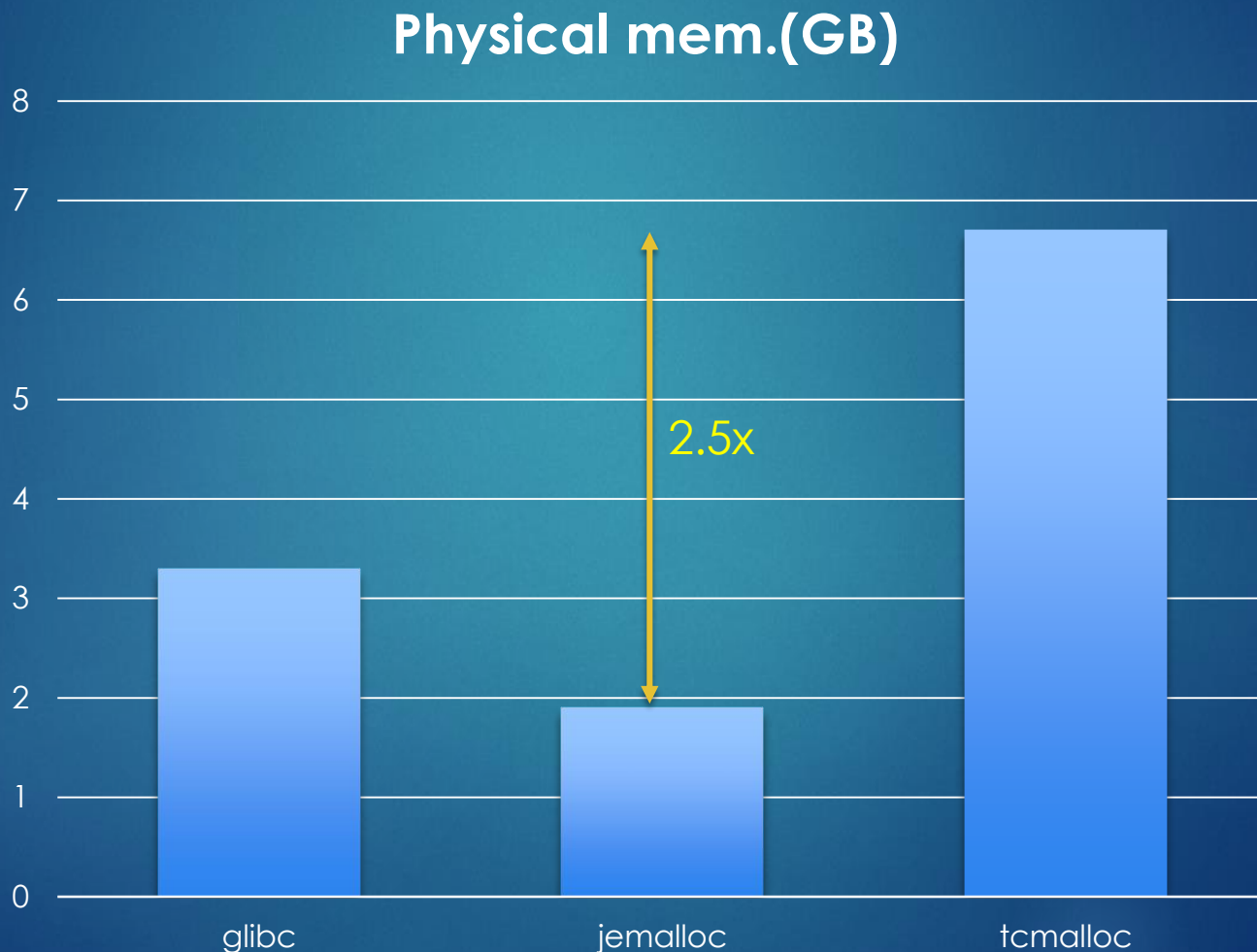


Eg: **>1M lines** C++ simulation.
On **128 cores / 16 NUMA CPUs**



Same about **memory consumption**
on 12 cores

6



Tool 1 : MALT

7

- ▶ **Memory management** can have **huge impact**
- ▶ Tool to **track mallocs**
- ▶ Report **properties** onto **annotated sources**

- ▶ Same **idea** than **valgrind/kcachegrind**
 - ▶ Annotated sources
 - ▶ Annotated call graphs
 - ▶ + **Non additive metrics** (for inclusive costs, eg. lifetime)
 - ▶ + **Time charts**
 - ▶ + **Properties distribution (sizes....)**

Web based GUI

8

Inclusive/Exclusive

Metric selector

Per line annotation

MATT WebView

Summary Alloc sites Time analysis Stack Alloc sizes Help

Allocated mem. ▾

Search

- 28.4 KB __libc_start_main
- 28.4 KB _start
- 28.2 KB main
- 12.5 KB testMaxAlive()
- 6.9 KB recurseA(int)
- 6.3 KB testThreads()
- 1.0 KB funcB()
- 1.0 KB testRecurseIntervalA(l...
- 1.0 KB testRecurseIntervalB(l...
- 704.0 B funcC()
- 704.0 B testParallelWithRecur...
- 128.0 B OutOfMainAlloc
- 128.0 B __cxx_global_var_init1
- 128.0 B global constructors ke...
- 128.0 B __libc_csu_init

```
/home/svalat/Projects/matt/src/lib/tests/simple-case.cpp
704 B 53 int * ptr = new int[16];
54 *(char*)ptr = 'c';//required otherwise new compilers will remove malloc/free
55 delete [] ptr;
56 }
57
58 /***** FUNCTION *****/
59 void funcB()
60 {
61     void * ptr = malloc(32);
62     *(char*)ptr='c';//required otherwise new compilers will remove malloc/free
63     free(ptr);
64     funcC();
65 }
66
67 /***** FUNCTION *****/
68 void funcA()
69 {
70     void * ptr = malloc(16);
71     *(char*)ptr='c';//required otherwise new compilers will remove malloc/free
72     free(ptr);
73     funcB();
74 }
75
76 /***** FUNCTION *****/
77 void recurseA(int depth)
78 {
79     if (depth > 0)
80     {
81         void * ptr = malloc(64);
82         *(char*)ptr='c';//required otherwise new compilers will remove malloc/free
83         free(ptr);
84         recurseA(depth-1);
85     }
86 }
87
88 /***** FUNCTION *****/
```

Total :
Allocated memory : 96 B
Freed memory : 96 B
Max alive memory : 96
2 alloc : [32 B, 48 B, 64 B]
2 free : [32 B, 48 B, 64 B]
Lifetime : [41.3 K, 42.1 K, 42.9 K] (cycles)

Function	Metric
__start	96.0 B
__libc_start_main	96.0 B
main	96.0 B
funcA()	96.0 B
funcB()	96.0 B
malloc	32.0 B
funcC()	64.0 B

Symbols

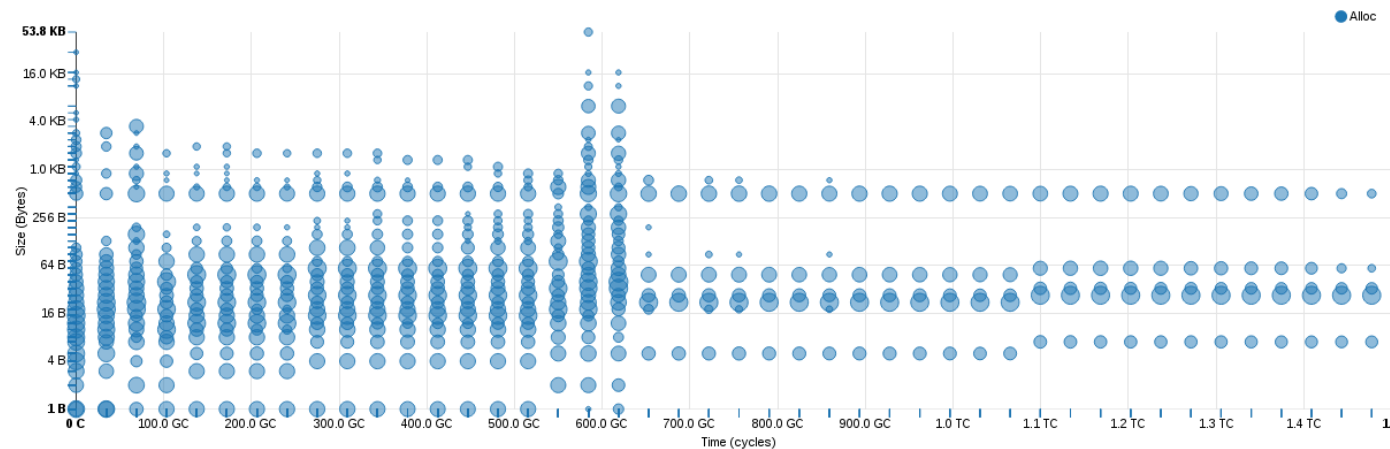
Details of symbol or line

Call stacks reaching the selected site.

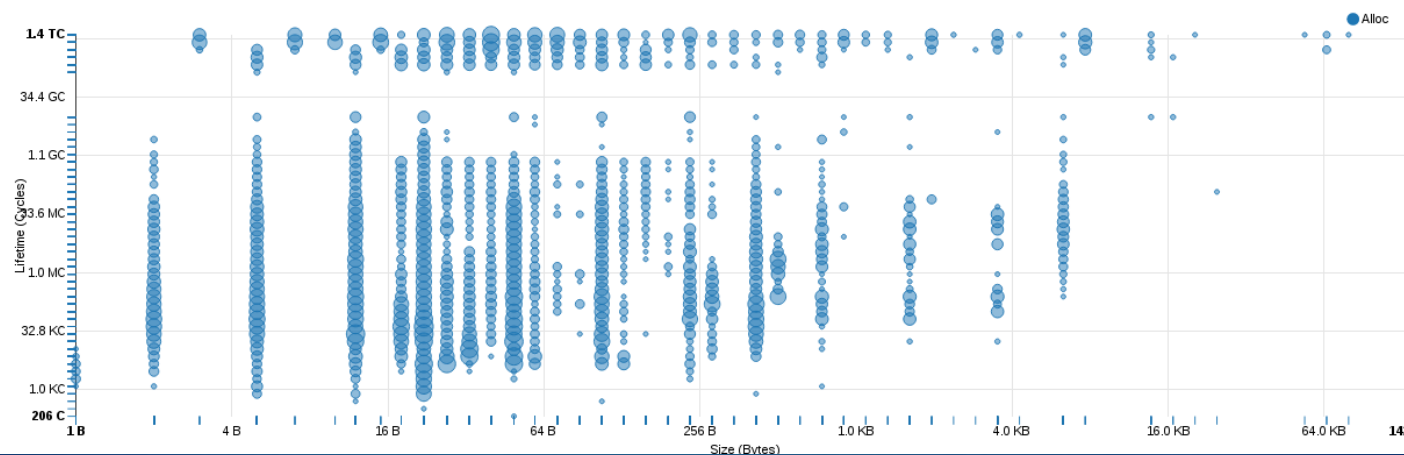
Example of time based view

9

Size over time



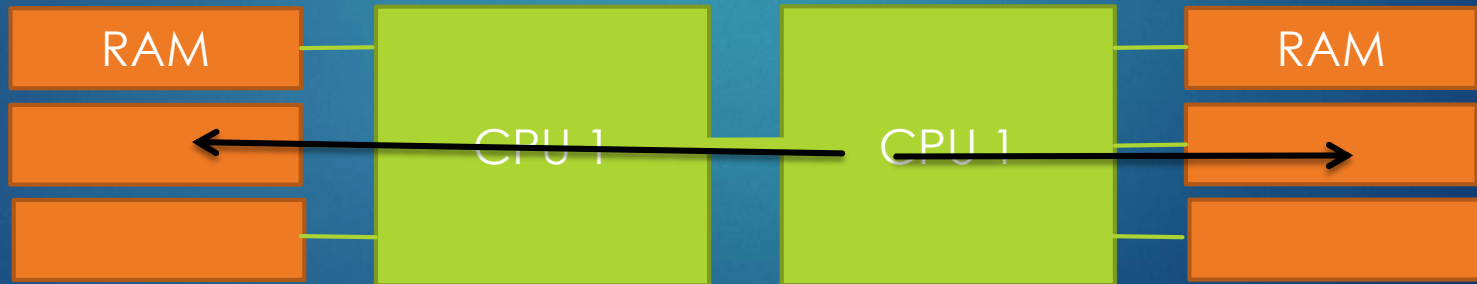
Lifetime over size



Tool 2 : NUMAPROF

10

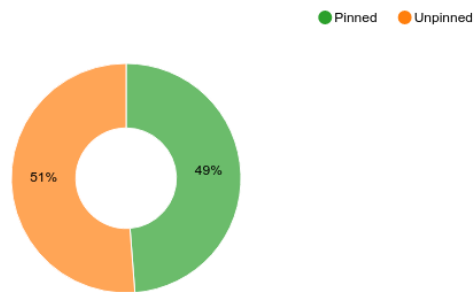
- ▶ Based on MALT code
- ▶ But about **NUMA**
- ▶ How to **detect remote** memory **accesses**
- ▶ Unsafe & **uncontrolled memory binding**



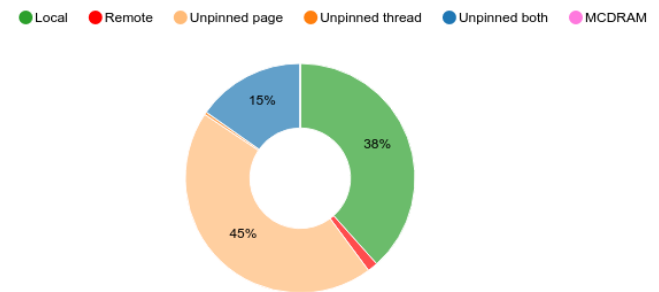
Some summary views

11

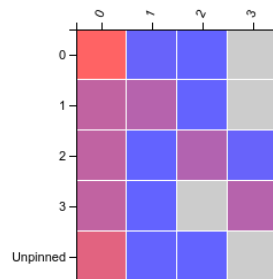
First touch



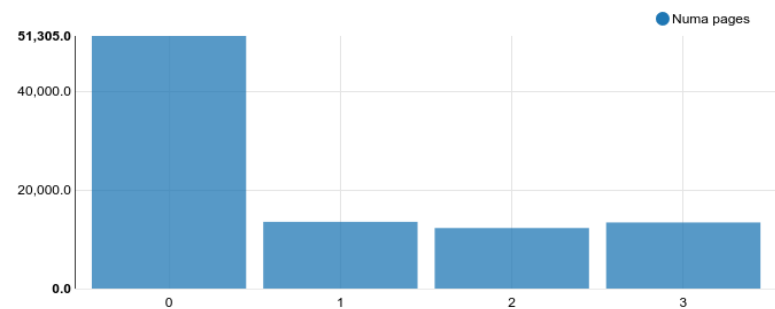
Memory access



Access matrix



Peak allocated numa pages



Still source annotation to understand code

12

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;
```


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



Short success

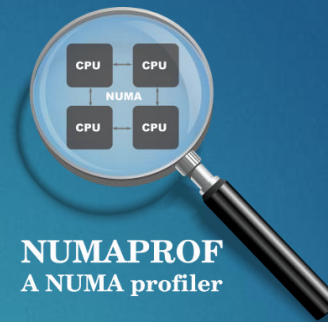
13

▶ MALT

- ▶ **20%** CPU saving on my **CERN 32 000** C++ code.
- ▶ Improvement on **2 commercial simulation** codes
- ▶ Profiled **CERN LHCb 1.5 million** line C++ code

▶ NUMAPROF

- ▶ **20% perf in 20 minutes** on 8000 lines simu.
- ▶ NUMA **Linux kernel policy bug** detected.
- ▶ CERN PhD. code **NUMA correctness**



Questions

Both tools under CeCILL-C on <http://memtt.github.io>

My researches : <http://svalat.github.io>

Example of success

MALT

- ▶ Reduce **CPU usage** of **30%** on the CERN app I was developing (mistake with C++11 `for(auto & it : lst)`)
32 000 C++ lines running on 500 servers.
- ▶ **Too large allocations** in a PhD. Student numerical simulation running on 500 cores while developing the tool.
- ▶ **Realloc pattern** in Fortran into **an industrial** R&D simulation code
- ▶ Unexpected **allocs generated** by GFortran **compiler** on another **industrial** R&D simulation **code**.
- ▶ Successfully ran on **CERN LHCb 1.5M lines** online analysis software

Example of success NUMAPROF

- ▶ **20% performance improvement** in 20 minutes on an unknown 8000 C++ lines simulation on Intel KNL
- ▶ **Linux Kernel bug** detected on NUMA management in conjunction with Transparent Huge Pages (while developing the tool).
Was detected at same time by other way by Red-Hat.... But.....
- ▶ **Confirmation** of NUMA **correctness** on a CERN/OpenLab PhD. Student code on Intel KNL