Kratos Profiling Experience

Carlos Roig
Riccardo Rossi **… @UPC & CIMNE**
Pooyan Dadvand

Denis Demidov **… @Supercomputer Center of Russian Academy of sciences**

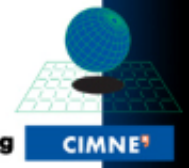
What is Kratos:

A large (2M+ lines of code) Finite Element general purpose program.

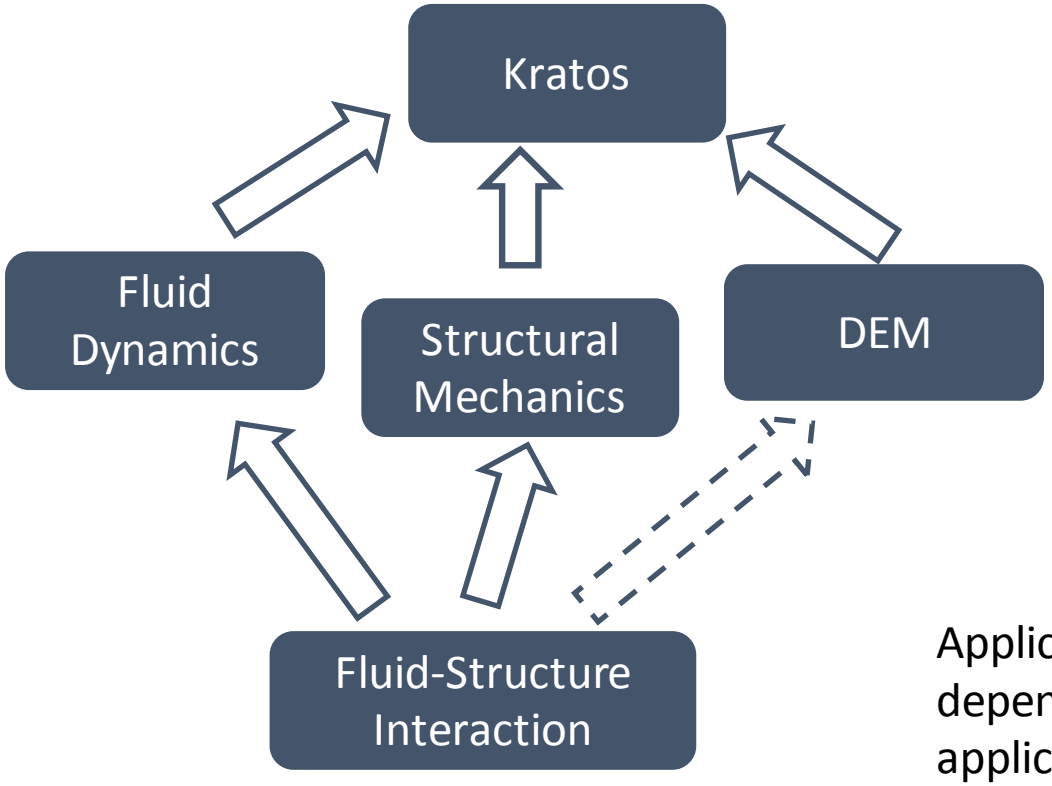
- BSD licensed
- **Python-driven** (computations are done in C++, so actually "mixed mode")
- Core & Applications approach → Each "application" is a shared library

Code can be found on GITHUB:

https://github.com/KratosMultiphysics/Kratos



Kratos Framework
Highly Modular Design

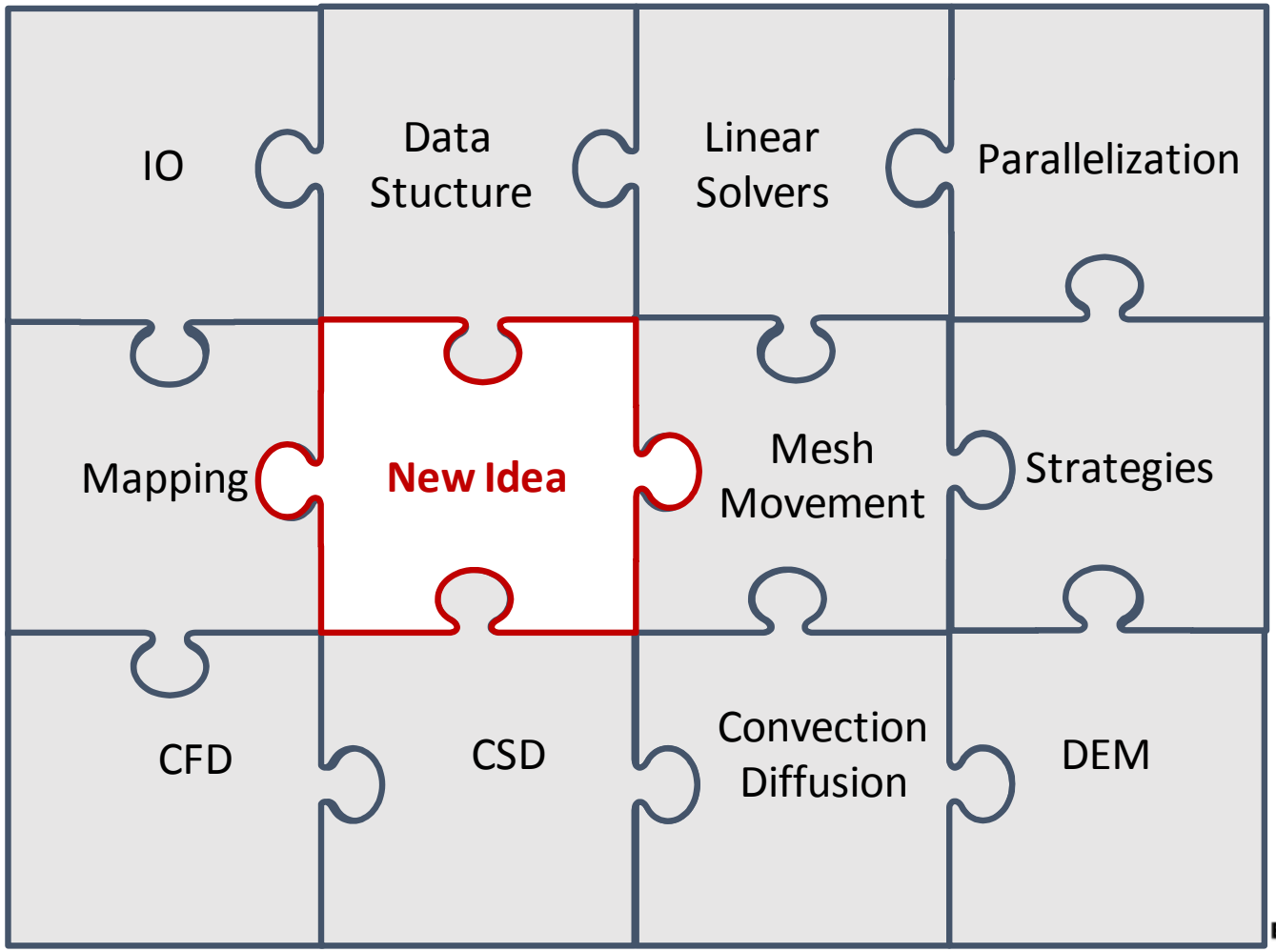


Numerical and programming core

Physics of the problem

Applications can depend on other applications

Motivation
Multi Disciplinary Implementation



What do we do here?

… we selected one application for which we had a very **bad scalability** and we applied @POP for help.

NOT our first attempt at profiling, but so far **only VTUNE** was working.

… and i think that any profiler out there was tried within the NUMEXAS Project …

What is the computational structure of our application?

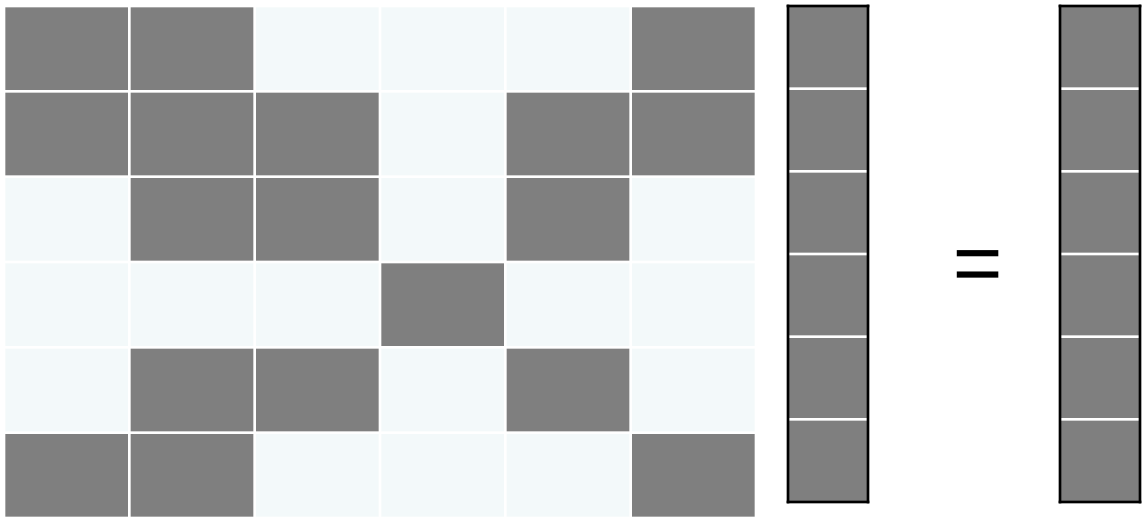
Two main phases.

1. "**BUILD**" phase (also known as FEM assembly phase)→where the physics is involved
2. "**SOLVE**" phase → a linear system is solved, by employing an AMG technique

BUILD PHASE

The goal is to fill a matrix of the type

$$\boldsymbol{Ax} = \boldsymbol{b}$$



Matrix is sparse
(block CSR structure)

How do we fill the matrix?

Starting from “zero”

0	0				0		x		0
0	0	0		0	0		y		0
	0	0		0			z	=	0
			0				w		0
	0	0		0			s		0
0	0				0		t		0

How do we fill the matrix?

Adding a matrix (full of 1s for the the sake of discussion) at the positions [0,1,5]

1	1				1		x		1
1	1	0		0	1		y		1
	0	0		0			z	=	0
			0				w		0
	0	0		0			s		0
1	1				1		t		1

How do we fill the matrix?

Adding a matrix (full of 2s for the the sake of discussion) at the positions [1,2,4]

1	1				1		x		1
1	1+2		2		2	1	y		1+2
		2	2		2		z	=	2
				0			w		0
		2	2		2		s		2
1	1					1	t		1

How do we fill the matrix?

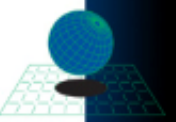
Adding a matrix (full of 3s for the the sake of discussion) at the positions [3]

1	1				1		x		1
1	1+2		2		2	1		y		1+2	
		2	2		2			z	=		2
				3				w			3
		2	2		2			s			2
1	1					1		t			1

Where is the parallelism?

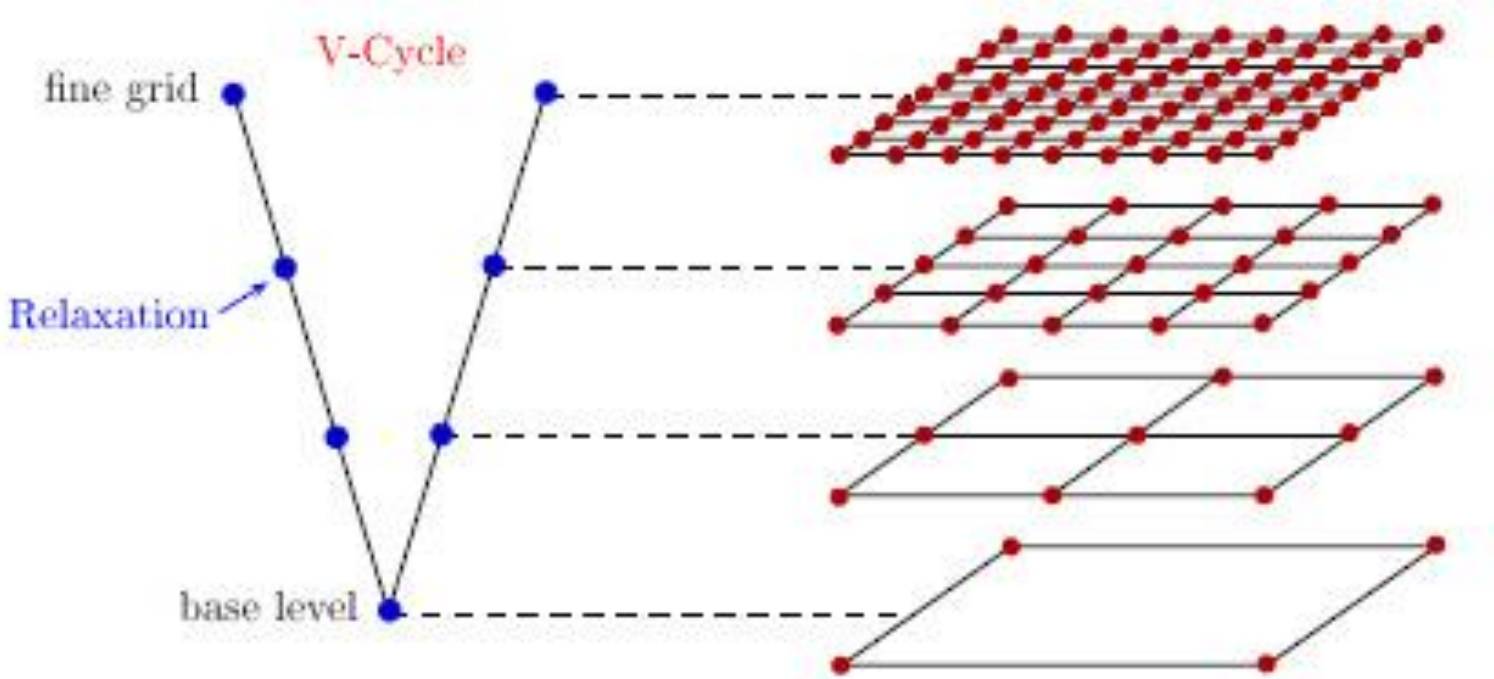
The finite element method is all about computing the "local matrices". They are NOT full of 1s!!

- The computation of every "local" matrix is embarassingly parallel (it requires gathering data to a local scratch space)
- HOWEVER **when we sum up the matrices we have potential clashes**.
- The destination matrix is sparse → A(i,j) requires a search in the CSR arrays!
- Sparse Matrix is composed of 4 by 4 blocks



AMG IDEA:

Do a “coarsening” **on the basis of the Provided matrix entries**



- Strongly scalable
- Weak scalability is difficult

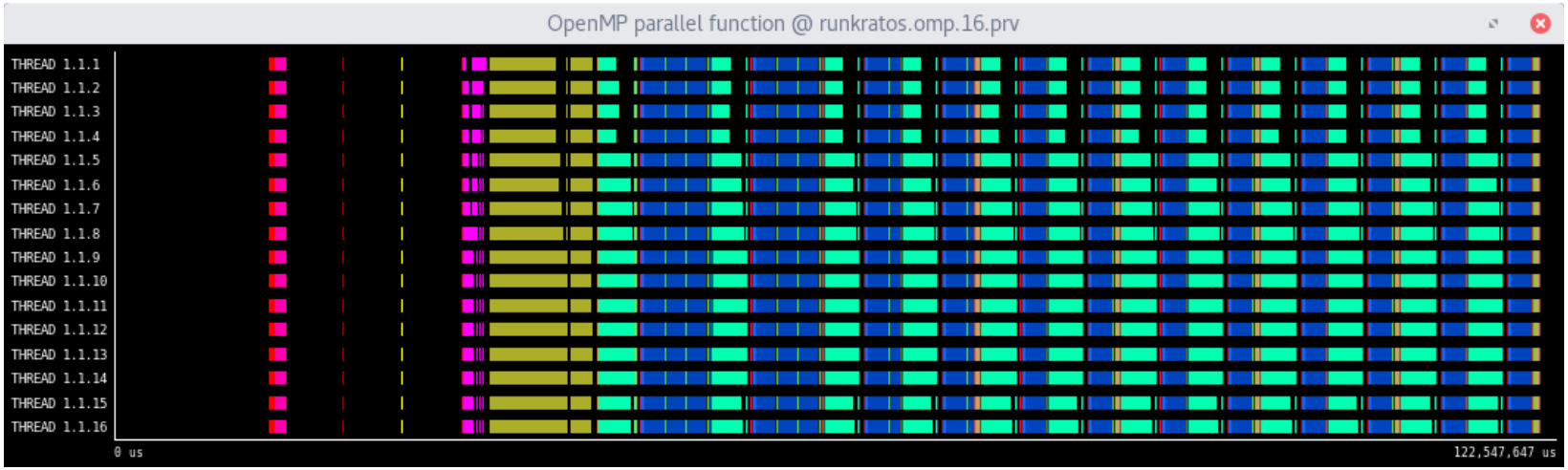
AMGCL library

In current presentation we employ **AMGCL** library:

https://github.com/ddemidov/amgcl

Optimization of the lib is part of what is presented here

STARTING POINT

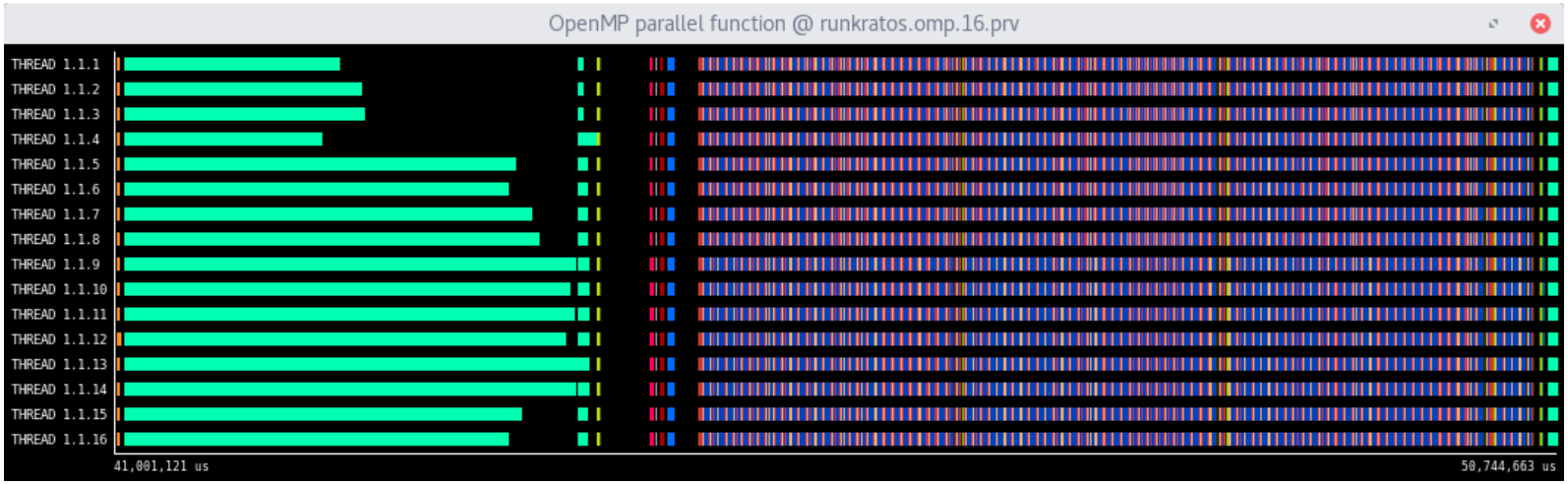


read

compute

Initialize System Matrix

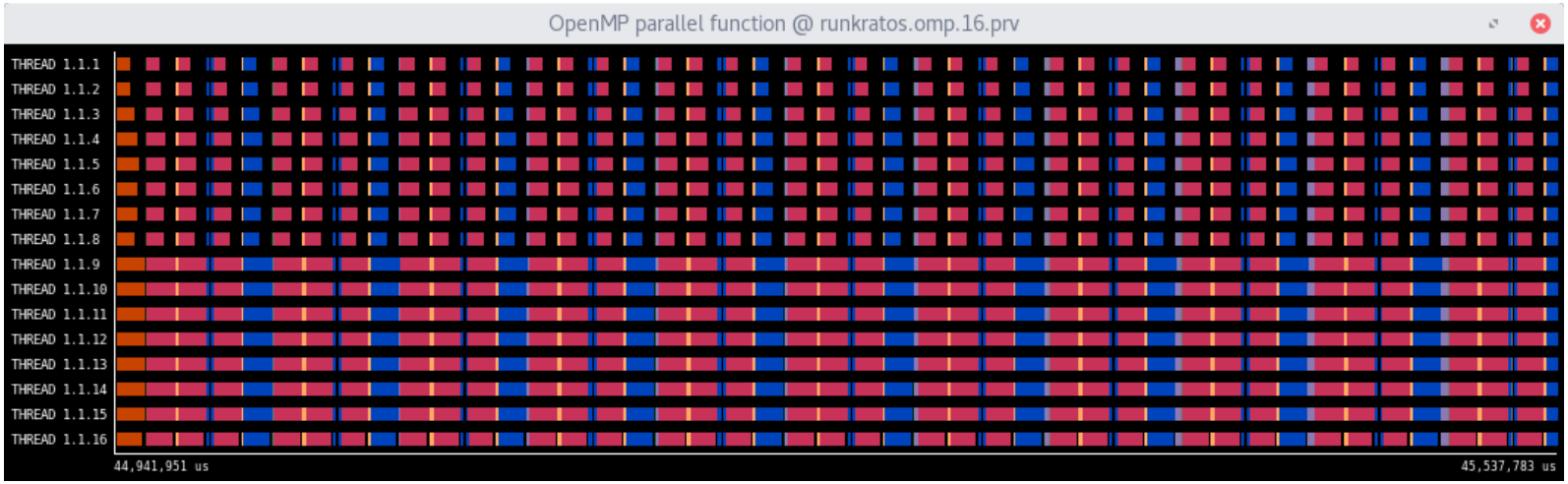
ZOOMING IN (ON COMPUTE)



build

Solve

ZOOMING IN (ON SOLVE)



Solve

ISSUES IDENTIFIED (thx @POP)

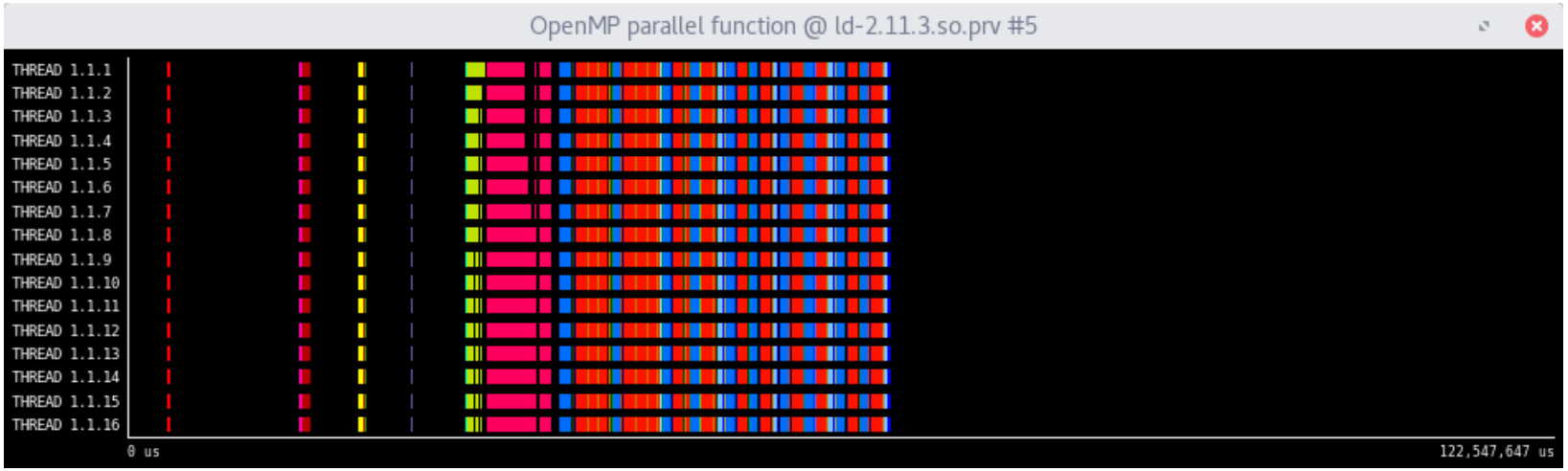
- Low IPC (falling out of a Cliff when stepping out from 8 to 16 cores). Bad cache locality? → IMPROVE NUMBERING
- Some NUMA effects →IMPROVE FIRST TOUCHING
- Loop Unbalance issues →LOOP SCHEDULING ?

Addressing NUMA

On the solver side, **problem identified on std::vector implementation** effectively preventing it → changed to own vector implementation

On the “build” phase, careful allocation of objects

IMPROVED VERSION



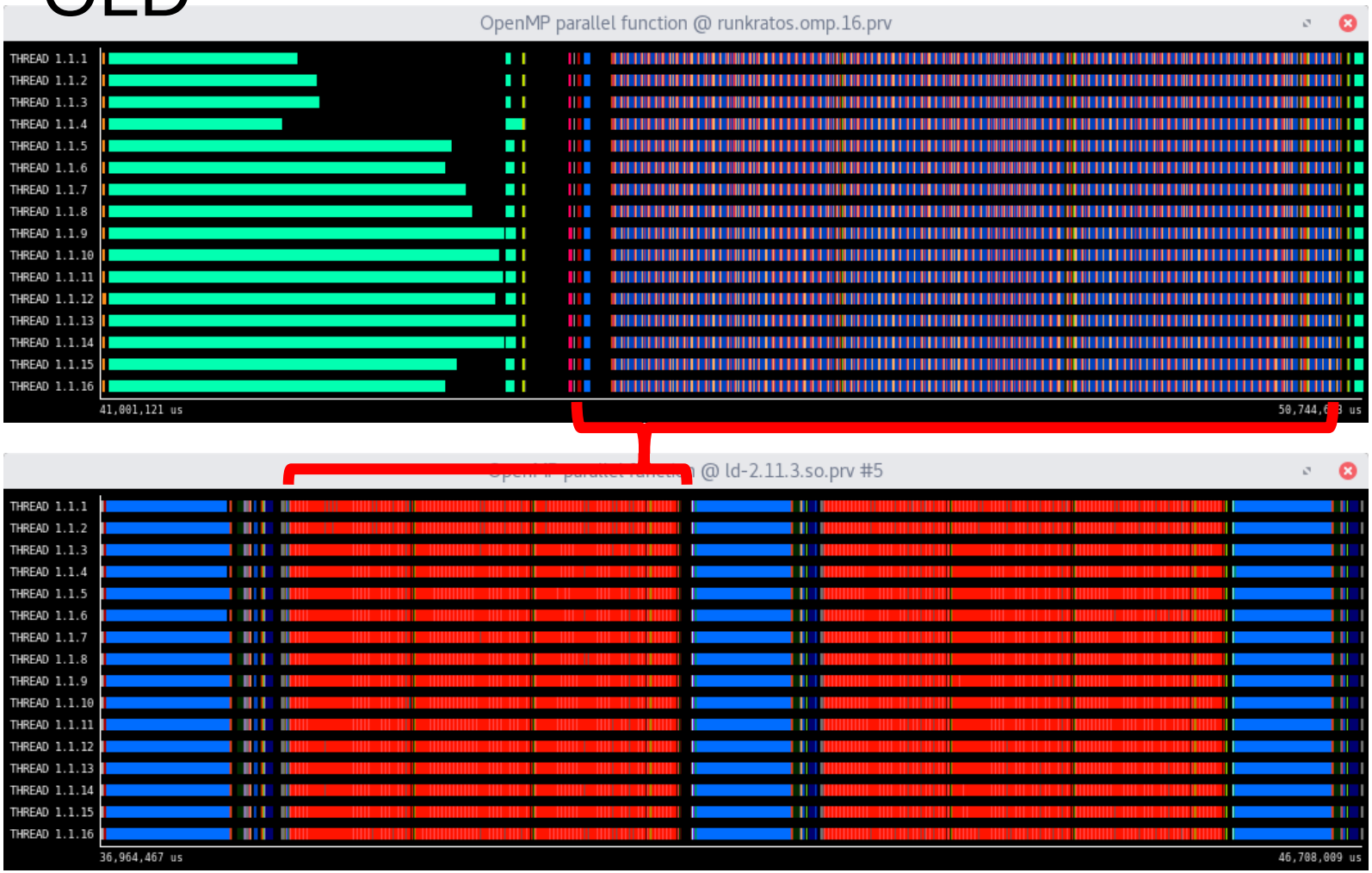
read

compute

Initialize System Matrix

ZOOMING IN (ON COMPUTE)

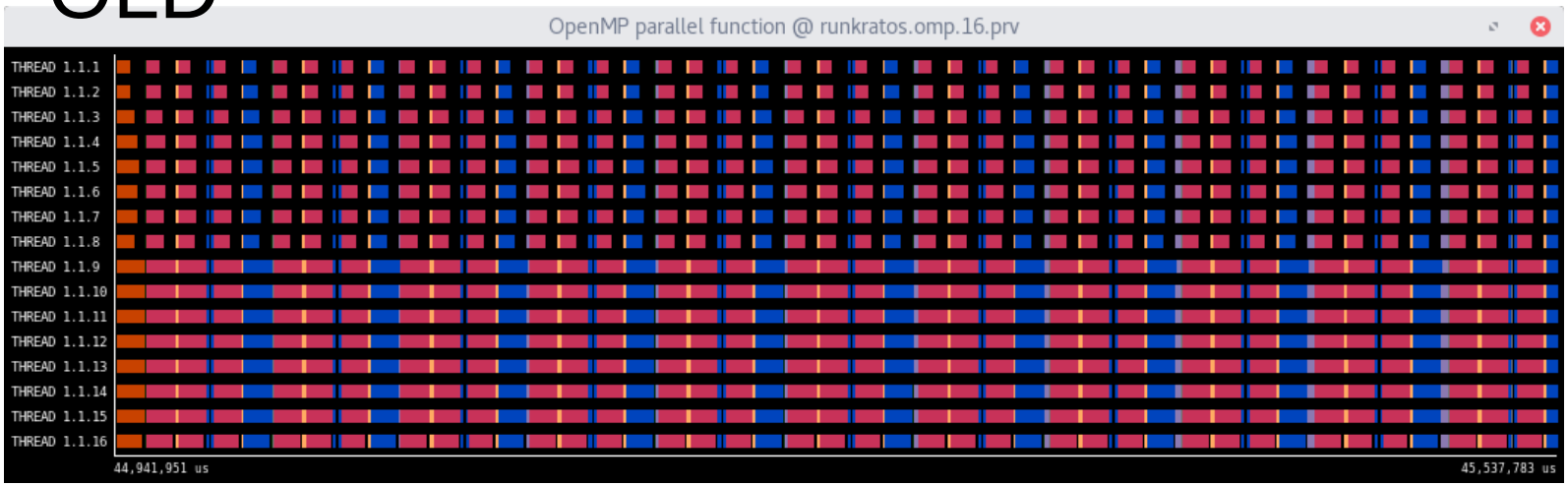
OLD

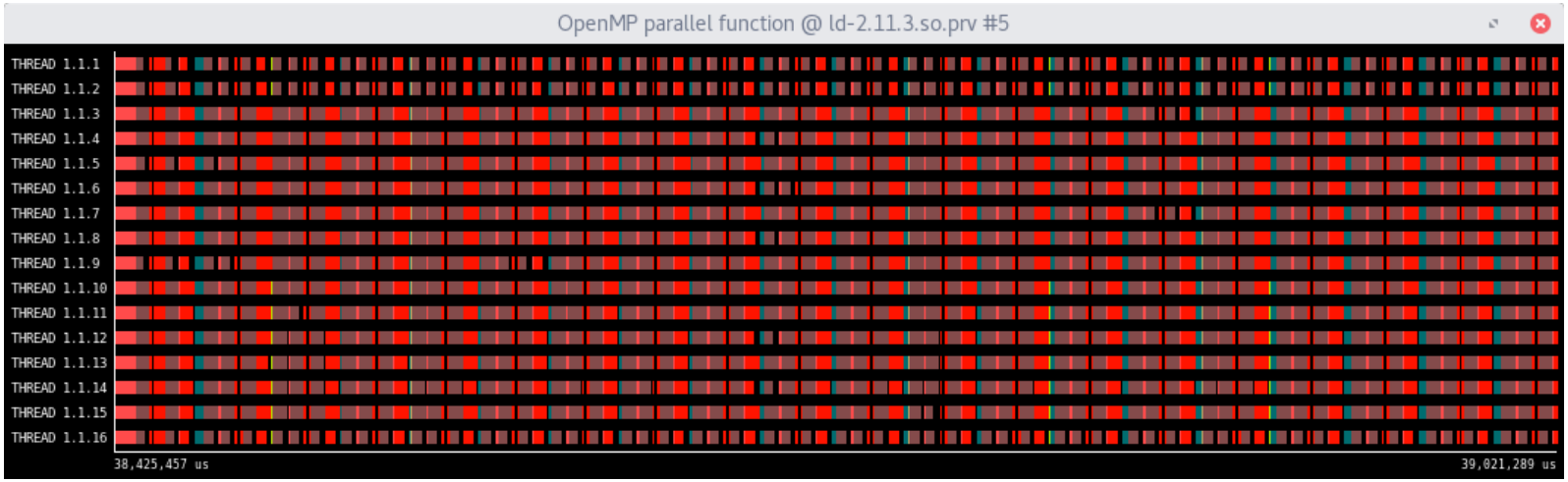


NEW

ZOOMING IN (ON SOLVE)

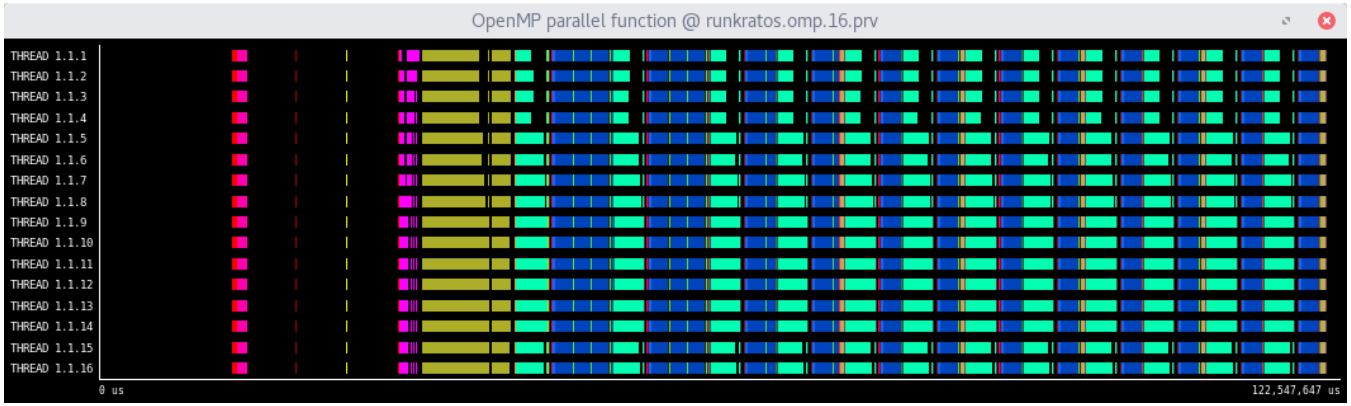
OLD



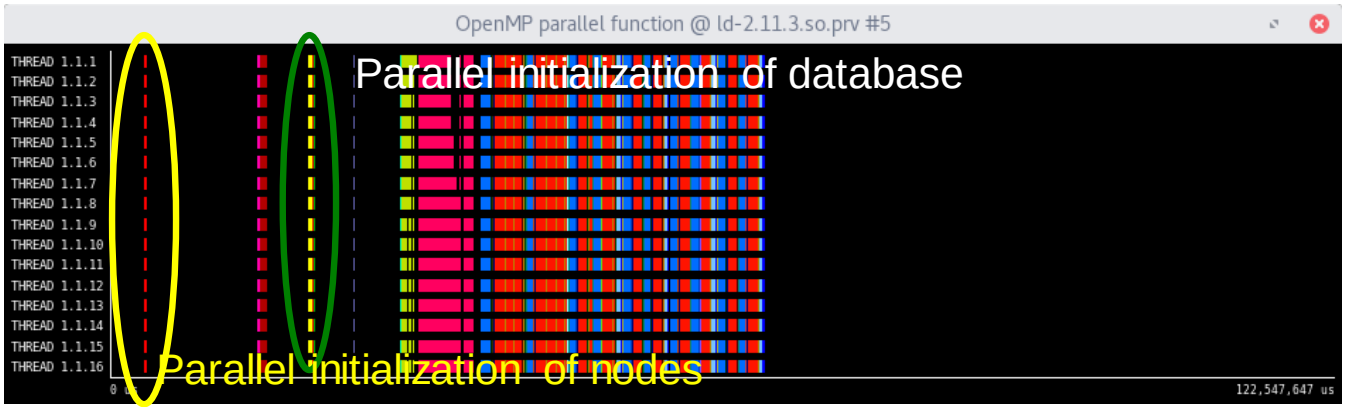


NEW

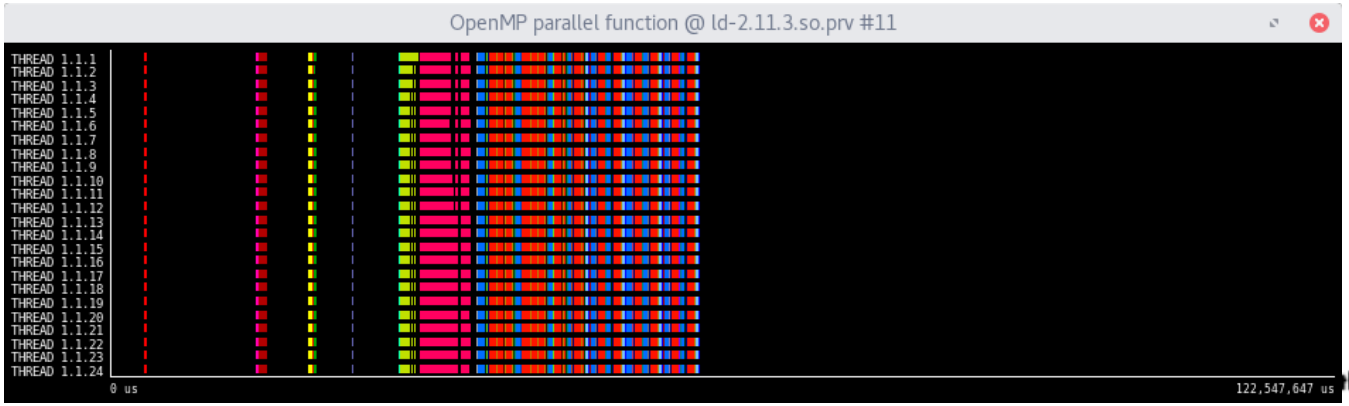
Overall conclusion



16 threads



16 threads



24 threads

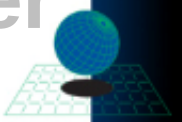
TODOs:

- RENUMBERING FOR OPTIMAL CACHE EFFICIENCY → early benchmarks show 20-30% further improvement in system solve
- SOME MORE PARALLEL INITIALIZATION

QUESTION:

Any hope to have Access to a KNL?

Would love to check on a sys with higher bandwidth

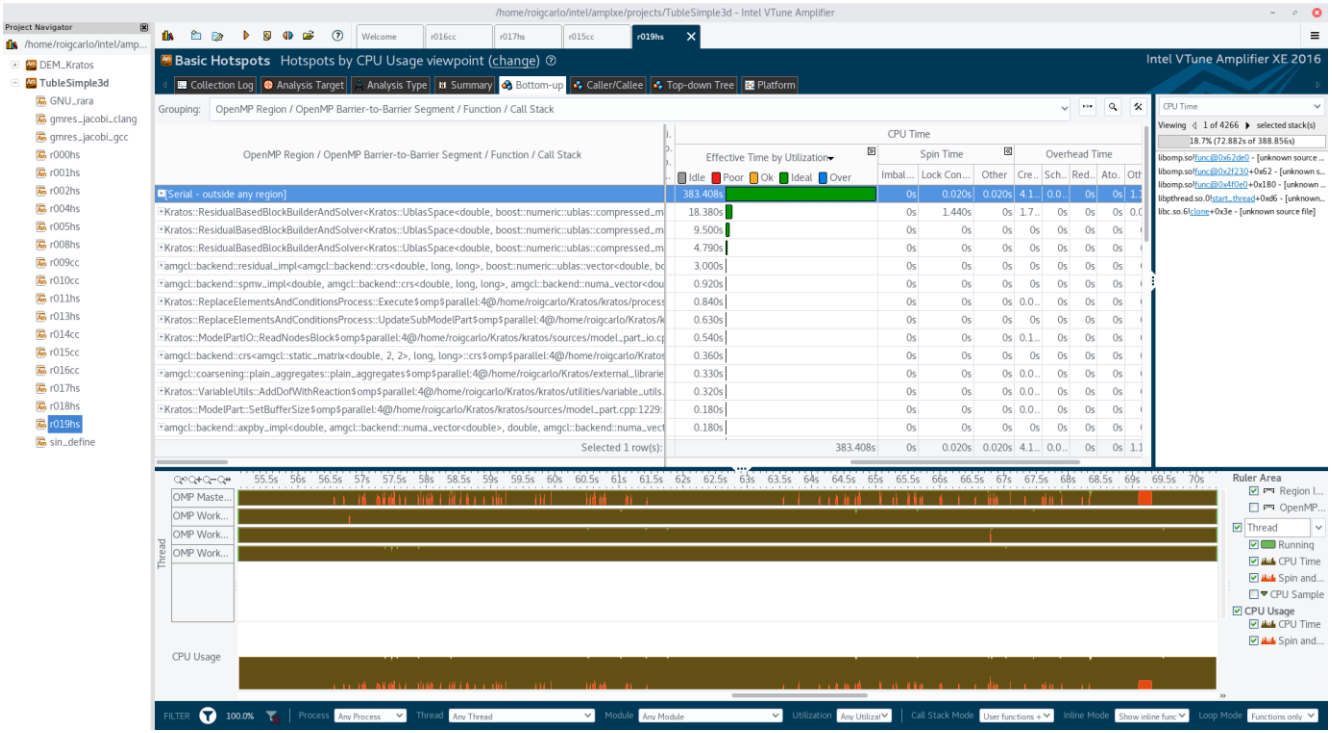


Suggestions:

No claim of being experts in using Extrae … so it may be our unexperience. However:

- Extrae better than VTUNE in highlighting parallel portions (as well as lack of parallelism)
- It would be useful to have indication of non-parallel parts (hard to say from within Extrae which portion of code is not in parallel –may be due to having a Python layer in the middle)
- VTUNE still superior in highlighting which portions of code are "hotter"
- Output could be nicer when a lot of templates are involved.
- Any hope to have information on the % of "local NUMA accesses" vs "remote ones"? – could even be slower than normal analysis, like valgrind
- Can we get info on what part of the code occupies more memory?
- It may be useful to package the software so that it is easy available on imporant machines? I am thinking e.g. of cray

Just a snapshot of VTUNE



- Portion of code running in parallel is visible
- Can easily zoom in in faction or even to lines of code
- When OMP_WAIT_POLICY=active everything appears to be in parallel **EVEN SERIAL REGIONS!!**

FEEDBACK @POP

- The tool itself provides some important insight. **The unbalance is very evident**
- Opportunity to speak with an expert is **VERY** important. Not obvious to diagnose results

…in next steps (if accepted in POP stage2) we'll try to take into account more of the early feedback. We'll also have an attempt including OmpSs, **we have however some limitations due to our need for portability, read … we need Windows!!**

On the OpenMP side…

- First touching approach not very convenient in the context of OO programming → why not making available a malloc-like function allowing to tell which processor should allocate data?
- How to force a given processor to run a task? (even serially)
- Affinity still an issue (for example the cray scheduler and the openmp runtime appear to “fight” each other.)

On the C++ standard side

It would be extremely convenient to have a std::vector constructor that allows first touching (changing the allocator is not enough, since it would imply zeroing twice the vector)