Performance POP up

EU H2020 Center of Excellence (CoE)

Performance Engineering for HPC:

Implementation, Processes & Case Studies

ISC 2017, Frankfurt, June 22nd 2017

POP CoE

- A **Center of Excellence**
  - On **Performance Optimization and Productivity**
  - Promoting **best practices in performance analysis and parallel programming**
- Providing **Services**
  - Precise understanding of application and system behavior
  - Suggestion/support on how to refactor code in the most productive way
- **Horizontal**
  - Transversal across application areas, platforms, scales
- For **academic AND industrial codes and users**

Partners

- **Who?**
  - BSC (coordinator), ES
  - HLRS, DE
  - JSC, DE
  - NAG, UK
  - RWTH Aachen, IT Center, DE
  - TERATEC, FR

A team with

- Excellence in performance tools and tuning
- Excellence in programming models and practices
- Research and development background AND
  proven commitment in application to real academic and industrial use cases

Motivation

Why?

- Complexity of machines and codes
  → Frequent lack of quantified understanding of actual behavior
  → Not clear most productive direction of code refactoring
- Important to maximize efficiency (performance, power) of compute intensive applications and the productivity of the development efforts

Target

- Parallel programs , mainly MPI /OpenMP … although can also look at CUDA, OpenCL, Python, …

3 levels of services

? Application Performance Audit

- Primary service
- Identify performance issues of customer code (at customer site)
- Small Effort (< 1 month)

! Application Performance Plan

- Follow-up on the service
- Identifies the root causes of the issues found and qualifies and quantifies approaches to address the issues
- Longer effort (1-3 months)

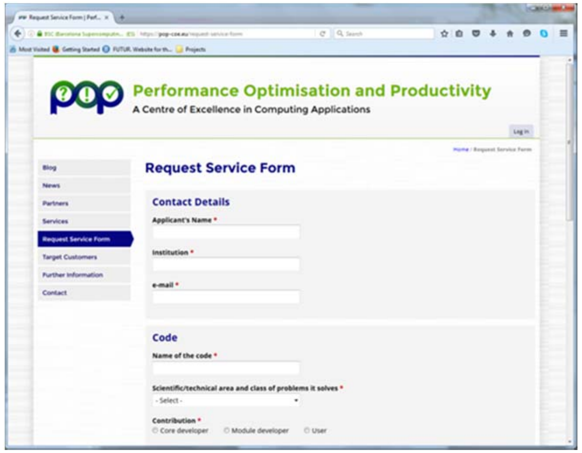
✓ Proof-of-Concept

- Experiments and mock-up tests for customer codes
- Kernel extraction, parallelization, mini-apps experiments to show effect of proposed optimizations
- 6 months effort

Reports

Software demonstrator

Apply @ http://www.pop-coe.eu



Target customers

- **Code developers**
  - Assessment of detailed actual behavior
  - Suggestion of more productive directions to refactor code
- **Users**
  - Assessment of achieved performance on specific production conditions
  - Possible improvements modifying environment setup
  - Evidences to interact with code provider
- **Infrastructure operators**
  - Assessment of achieved performance in production conditions
  - Possible improvements modifying environment setup
  - Information for allocation processes
  - Training of support staff
- **Vendors**
  - Benchmarking
  - Customer support
  - System dimensioning/design

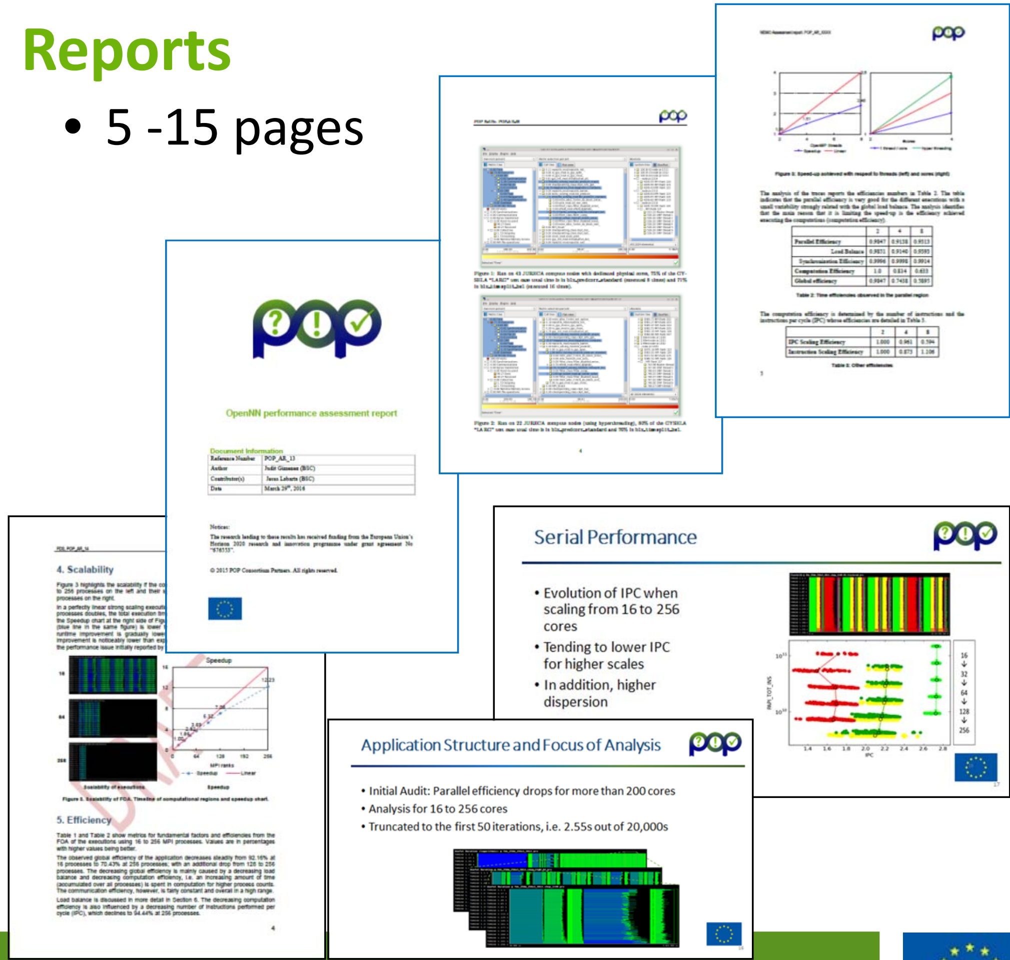
Activities (June 2017)

- **Services**
  - Completed/reporting: 80
  - Codes being analyzed: 21
  - Waiting user / New: 22
  - Cancelled: 10

- **By type**
  - Audits: 95
  - Plan: 15
  - Proof of concept: 13

+ 5 training workshops

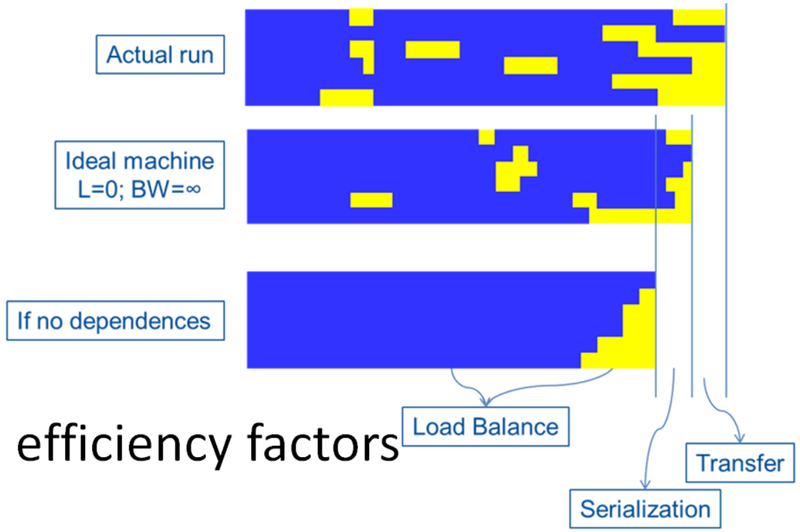
- **Reports**
  - 5 -15 pages



Fundamental performance factors

- Factors modeling parallel efficiency
  - **Load balance** (LB)
  - **Communication**
    - **Serialization** (or Micro load balance)
    - **Transfer**

$$\eta_{\parallel} = LB * \underbrace{Ser * Trf}_{CommEff}$$



- Factors describing serial behavior
  - Computational complexity: **#instr**
  - Performance: **IPC**
  - **Core frequency**
  - Actual values, scaling behavior, impact on parallel efficiency factors

M. Casas et al, “Automatic analysis of speedup of MPI applications”. ICS 2008.

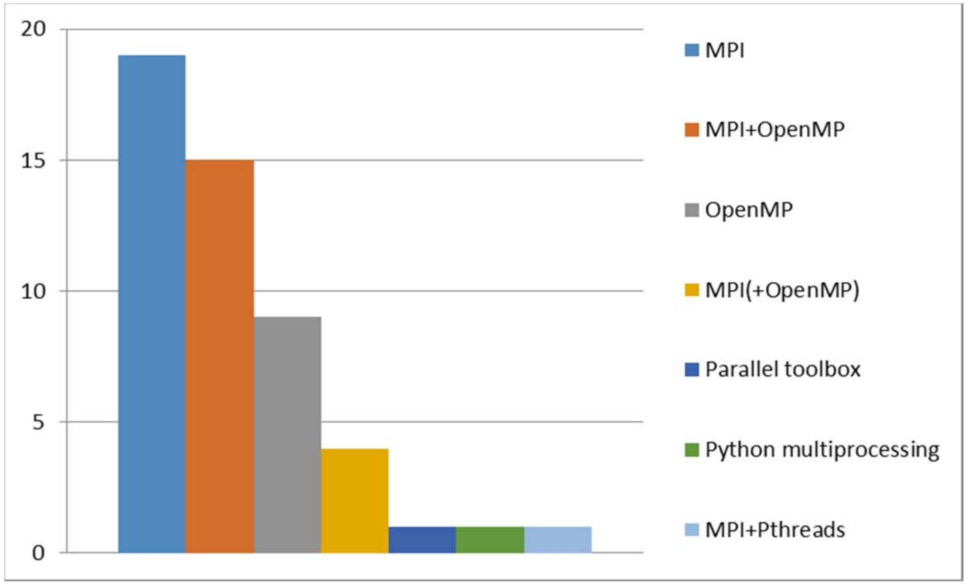
Efficiencies

	2	4	8	16
Parallel Efficiency	0.9834	0.9436	0.8980	0.8478
Load Balance	0.9871	0.9687	0.9099	0.9177
Serialization efficiency	0.9975	0.9770	0.9938	0.9395
Transfer Efficiency	0.9988	0.9970	0.9931	0.9833
Computation Efficiency	1.000	0.9590	0.8680	0.6953
Global efficiency	0.9834	0.9049	0.7795	0.5894

	2	**4**	**8**	**16**
IPC Scaling Efficiency	1.000	0.9932	0.9591	0.8421
Instruction Scaling Efficiency	1.000	0.9721	0.9393	0.9075
Core frequency efficiency	1.000	0.9932	0.9635	0.9098

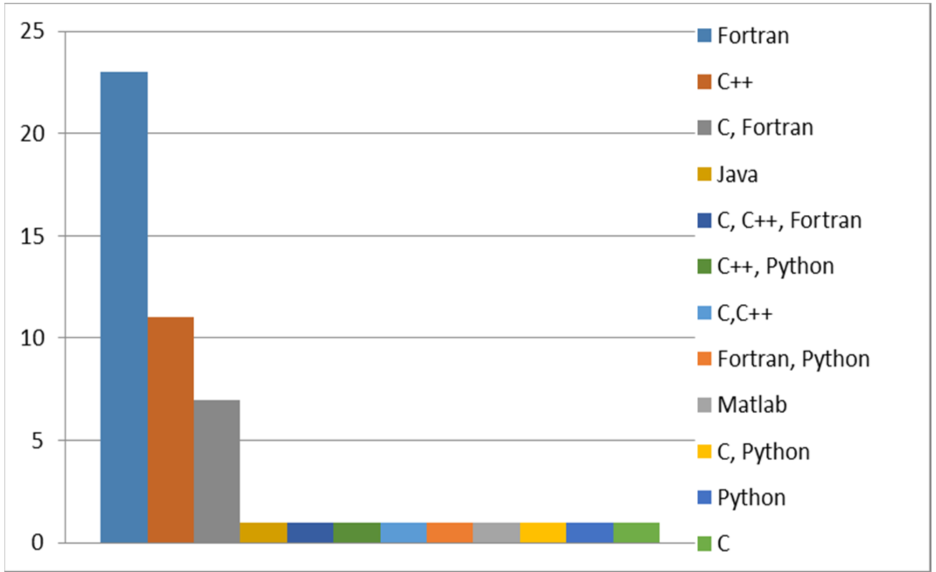
Audit characterization

Code



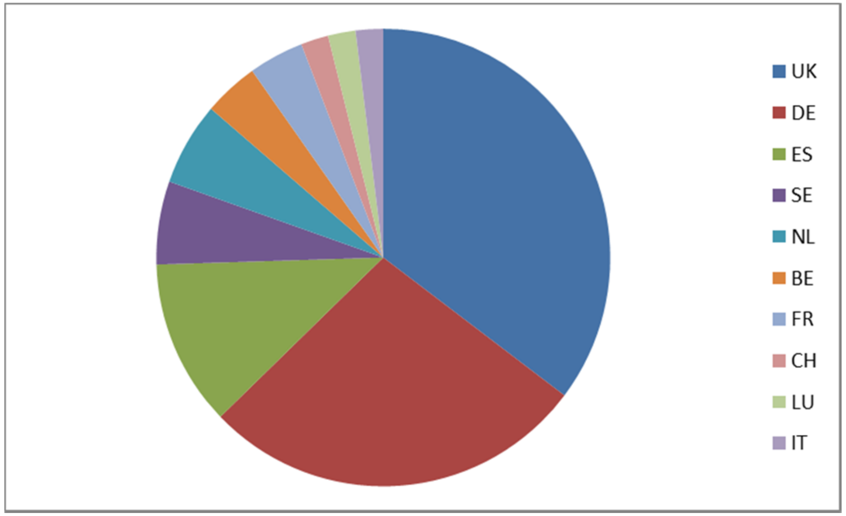
- **Parallel programming model**
  - 77% MPI or MPI+X
  - 17% pure OpenMP
  - Few from new paradigms

- **Programming language**
  - 64% Fortran (+X) as expected
  - 9.4% Python (+X) not that expected



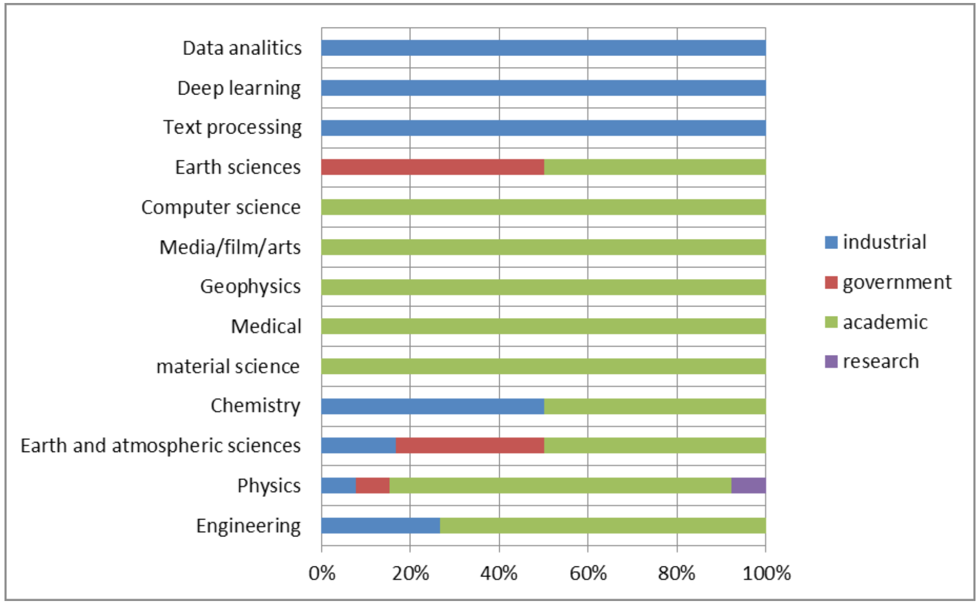
Audit characterization

User profile



- **Country**
  - 23% requests from countries outside the consortium
  - 33.9% UK, 26.3% DE, 13.2% ES, 3.6% FR

- **User institution versus code area**
  - Industrial companies provide all cases from new HPC sectors

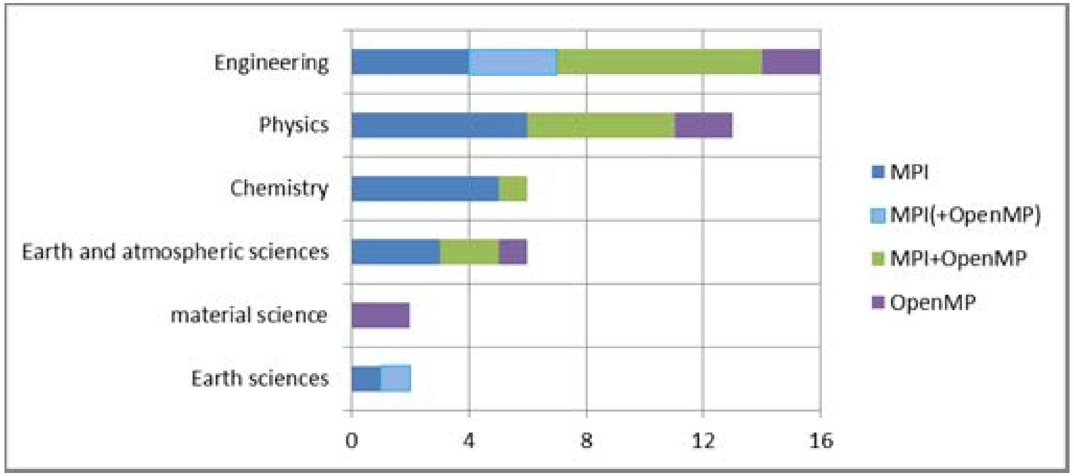


Audit characterization

Code

- **Scientific/technical area**
  - Dominated by Engineering and Physics
  - 90.5% of the requests from traditional HPC sectors
  - But also some requests on Data analytics, Deep learning, Medical, Media film, Text processing

Area versus parallel programing model



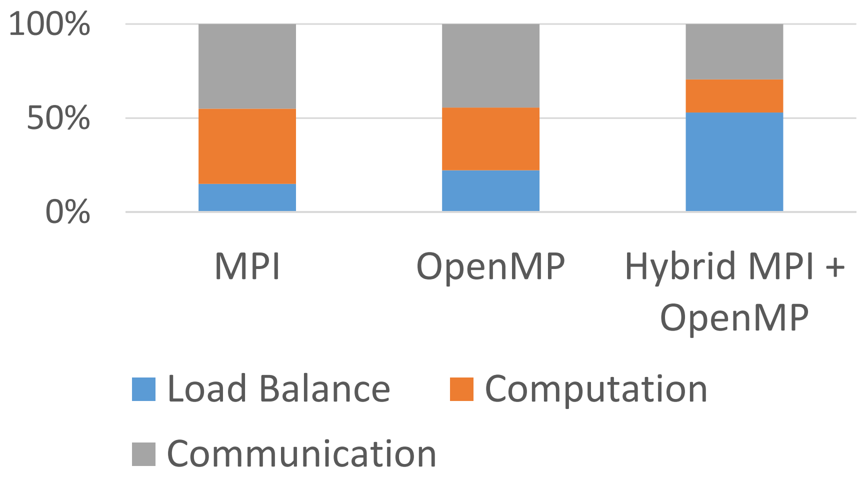
Other activities

- **Promotion and dissemination**
  - Market and community development
  - Dissemination material and events
- **Customer advocacy**
  - Gather customers feedback, ensure satisfaction, steer activities
- **Sustainability**
  - Explore business models
- **Training**
  - Best practices on the use of the tools and programming models
    - Cooperation with other CoEs (EoCoE)
    - Lot of interest … customers want to learn how to do it themselves

Audit characterization

Performance Audit results

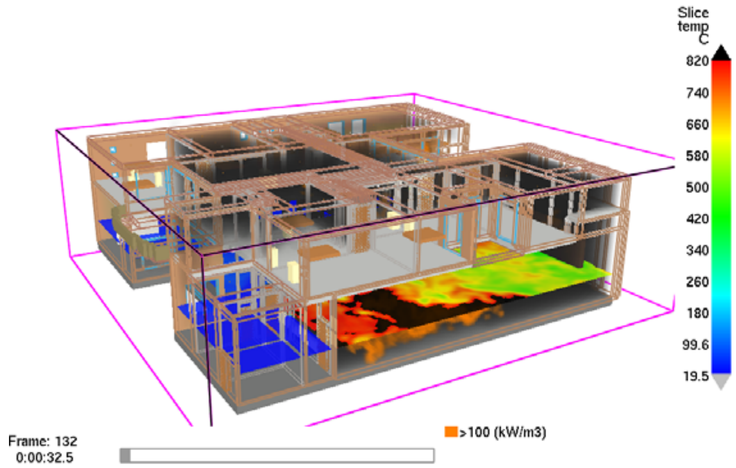
- **Parallel efficiency**
  - At least 67% would benefit / require optimizations (acceptable + bad)
  - Most frequent reason for acceptable efficiency is data transfer and for bad efficiency is load balance (+ data transfer)



- **Serial performance (IPC)**
  - 44% have IPC >1 for all regions
  - Others may benefit from a serial performance improvement
    - 24% general IPC < 1

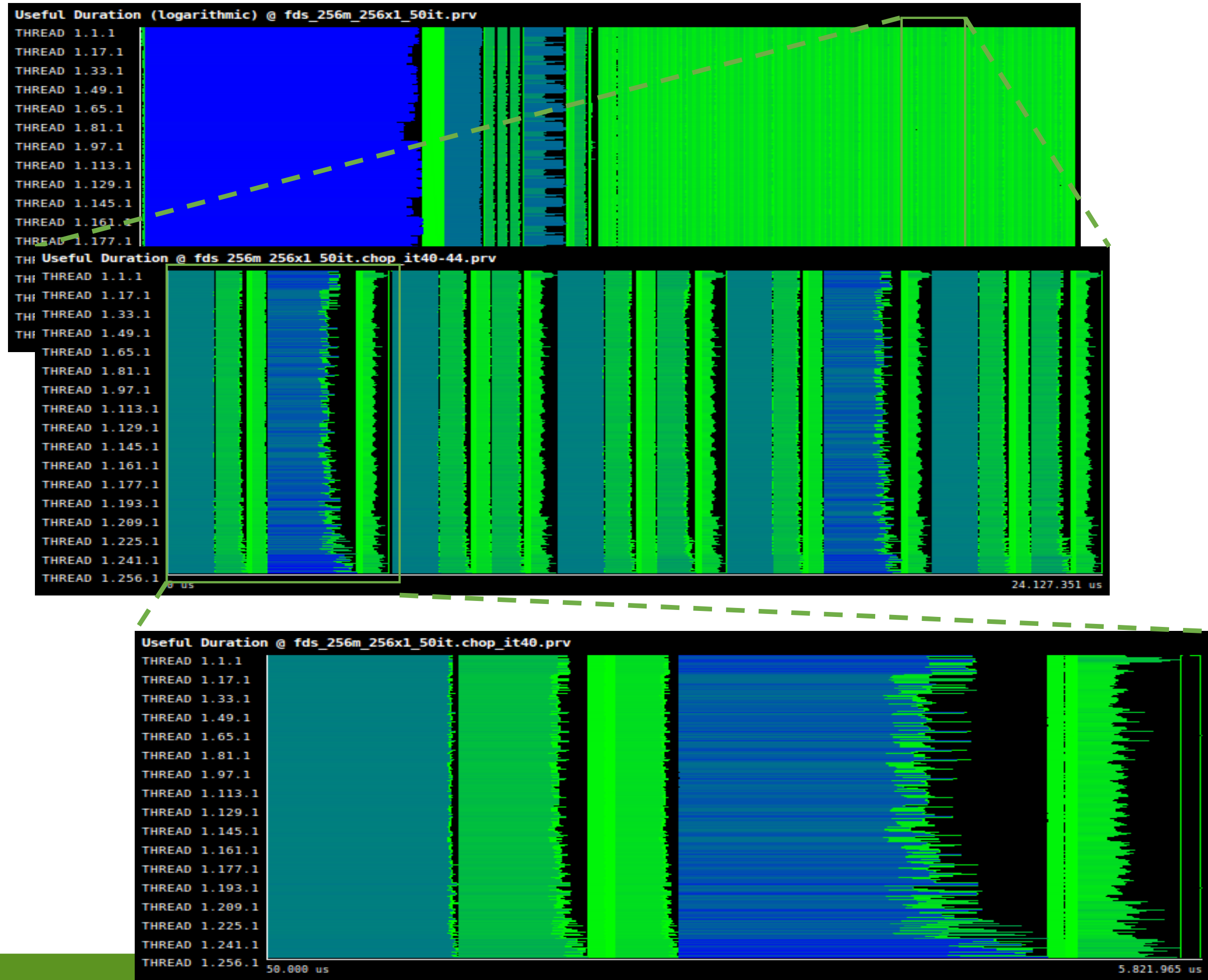
Case study: FDS Audit

- Customer:
  - SME
  - User of the code
- Code: FDS (Fire dynamics simulation)
  - Simulates fire and smoke development in structures
- Code Area: Engineering
- Performance Audit:
  - Efficiency drop above 200 cores
  - Evaluate efficiency running @ MareNostrum



Spatio-temporal structure

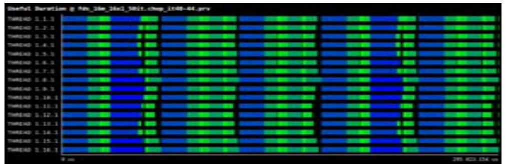
- Initialization
- Iterative phase



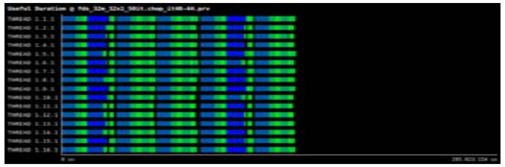
Scalability

4 iterations

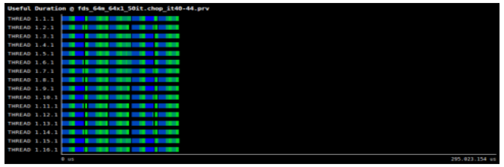
16



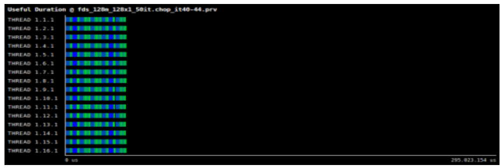
32



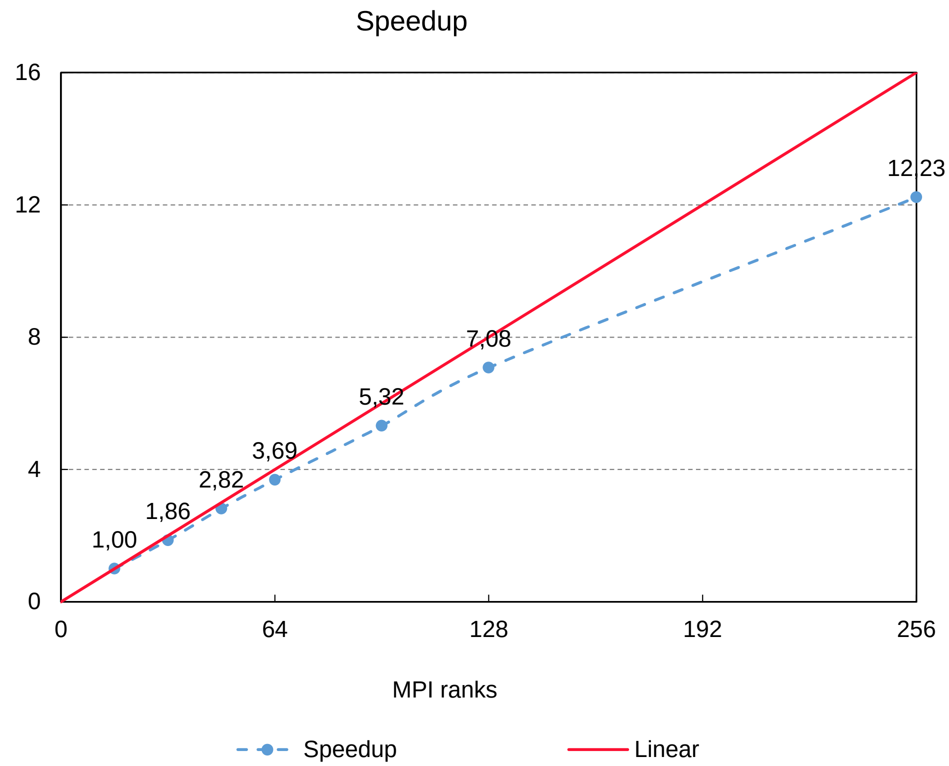
64



128



256



Efficiency

	32	48	64	96	128	256
Parallel Efficiency	91.74%	90.56%	88.74%	84.66%	86.41%	78.95%
Load Balance	94.60%	92.49%	93.40%	85.84%	87.05%	81.32%
Comm. Efficiency	96.97%	97.92%	95.01%	98.63%	99.26%	97.08%
Serialization	96.99%	97.95%	95.05%	98.70%	99.37%	97.54%
Transfer	99.98%	99.97%	99.96%	99.93%	99.89%	99.53%
Computation Scalability*	100.00%	102.51%	102.60%	103.55%	101.17%	95.64%
Global Efficiency	91.74%	92.84%	91.05%	87.67%	87.42%	75.50%

Table 1. Time efficiencies for the FOA from executions using 16 to 256 processes.

	32	48	64	96	128	256
IPC Scalability*	100.00%	101.33%	101.33%	101.33%	100.44%	98.22%
Instructions Scalability*	100.00%	101.34%	102.02%	101.90%	100.85%	97.71%

Table 2. Other efficiencies for the FOA from executions using 16 to 256 processes.

* Reference values are useful computation, IPC and total instructions based on 32 ranks.

Efficiency

	32	48	64	96	128	256
Parallel Efficiency	91.74%	90.56%	88.74%	84.66%	86.41%	78.95%
Load Balance	94.60%	92.49%	93.40%	85.84%	87.05%	81.32%
Comm. Efficiency	96.97%	97.92%	95.01%	98.63%	99.26%	97.08%
Serialization	96.99%	97.95%	95.05%	98.70%	99.37%	97.54%
Transfer	99.98%	99.97%	99.96%	99.93%	99.89%	99.53%
Computation Scalability*	100.00%	102.51%	102.60%	103.55%	101.17%	95.64%
Global Efficiency	91.74%	92.84%	91.05%	87.67%	87.42%	75.50%

Table 1. Time efficiencies for the FOA from executions using 16 to 256 processes.

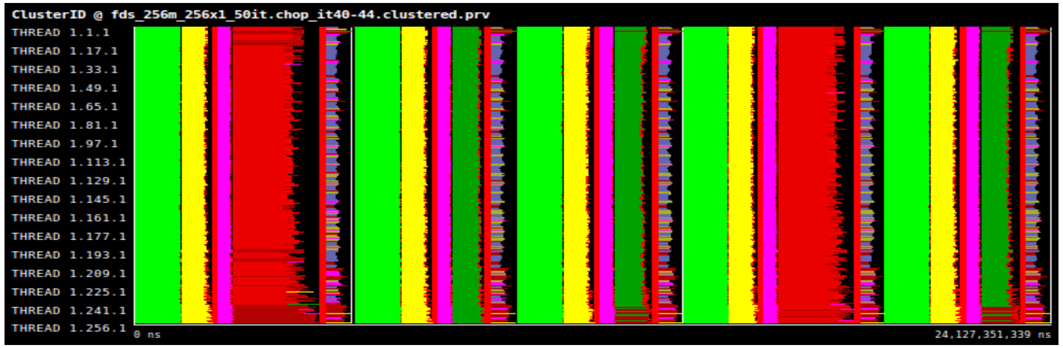
	32	48	64	96	128	256
IPC Scalability*	100.00%	101.33%	101.33%	101.33%	100.44%	98.22%
Instructions Scalability*	100.00%	101.34%	102.02%	101.90%	100.85%	97.71%

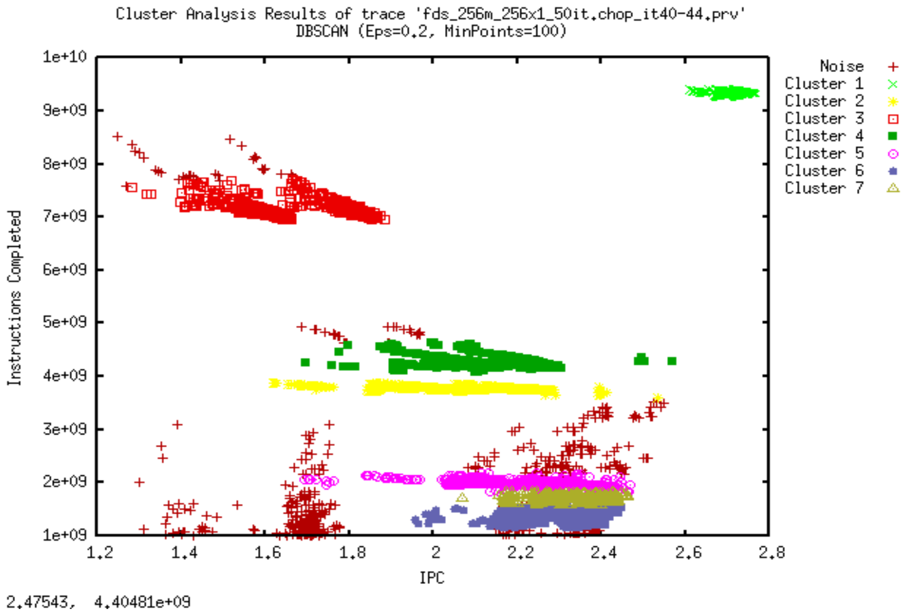
Table 2. Other efficiencies for the FOA from executions using 16 to 256 processes.

* Reference values are useful computation, IPC and total instructions based on 32 ranks.

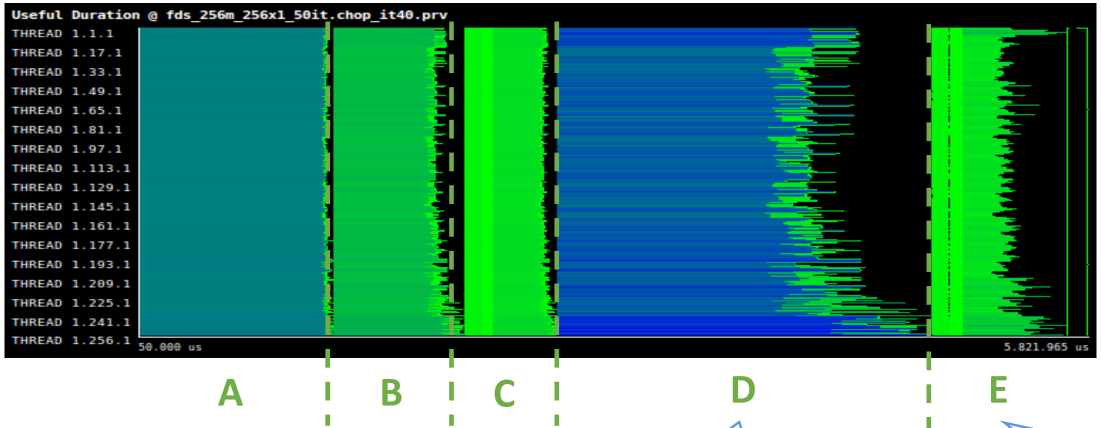
More on structure → clustering

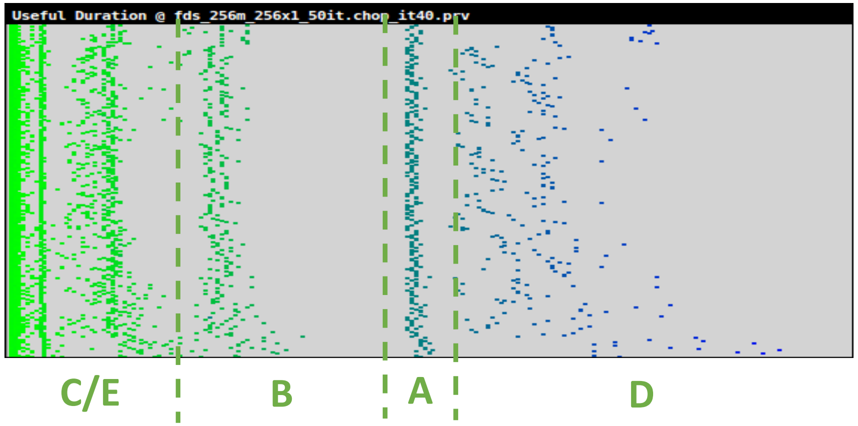
- Structure
  - Different behaviour every fourth iteration
  - Different behaviours at the first and last ranks in some phases ?
- Sequential performance insight
  - Imbalance in instructions **and** IPC accumulate
  - Variability in IPC





Load Balance – Main Conttributors





- *DUMP_BNDF* (dump.f90:7075)

- Two loops within *RADIATION_FVM* (radi.f90:611) beginning at line 1113 and 1177
- *DIVERGENCE_PART_1* (divg.f90:14) and its subroutine *SPECIES_ADVECTION* (difg.f90:857).

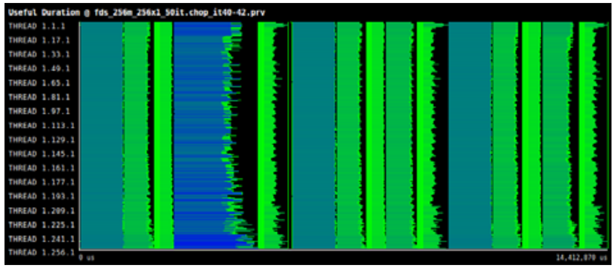
Refactoring ?

- Techniques
  - Taskify + DLB?
  - Balance IPC ?
  - Domain decomposition?
  - …
- Within reach, interest, … of customer ?

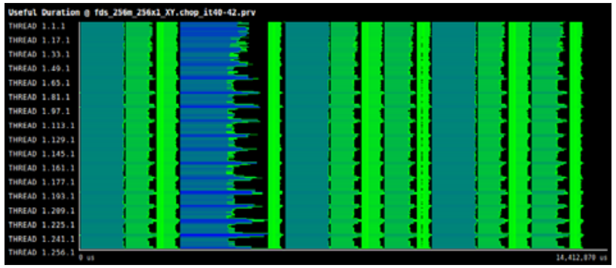
Refactoring ?

- Techniques
  - Taskify + DLB?
  - Balance IPC ?
  - **Domain decomposition?**
  - …
- Within reach, interest, … of customer?

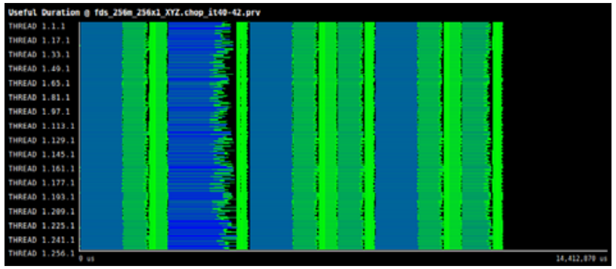
Decomposition: X
Load balance: 80%



Decomposition: XY
Load balance: 81%
Rel. runtime: 95%

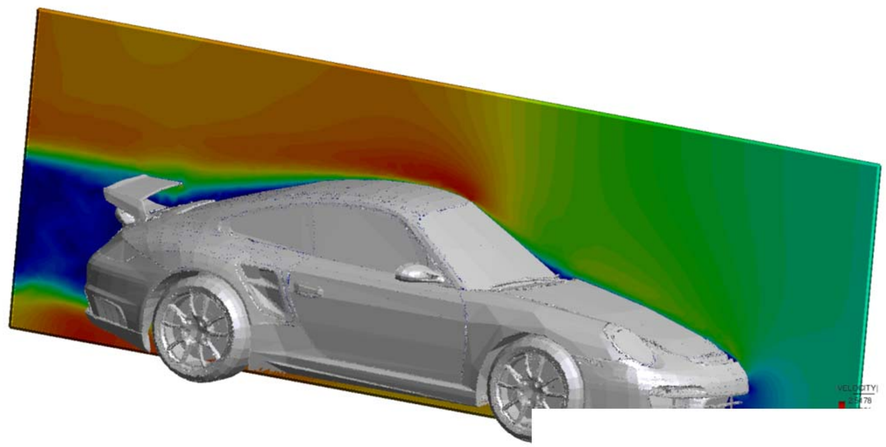


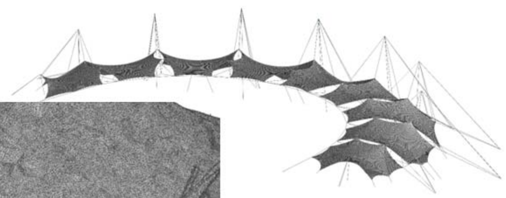
Decomposition: XYZ
Load balance: 91%
Rel. runtime: 80%

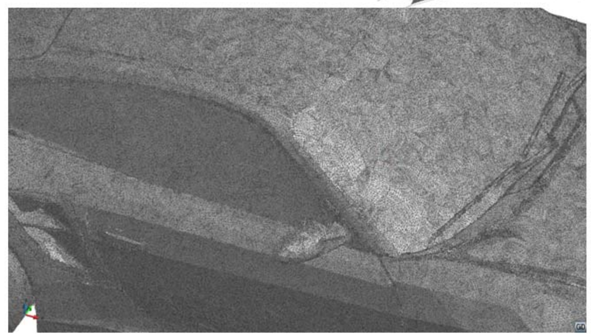


Case study: Kratos

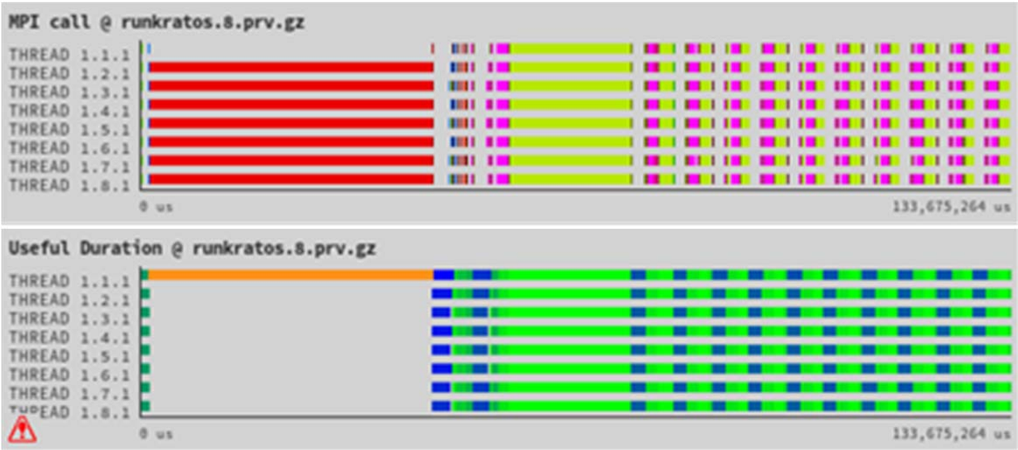
- Customer:
  - Research center
  - Developer of the code
- Code: Kratos
  - Multi-physics FE
- Code Area: engineering
- Performance Audit:
  - Happy with MPI scaling
  - Concerns on OpenMP scaling

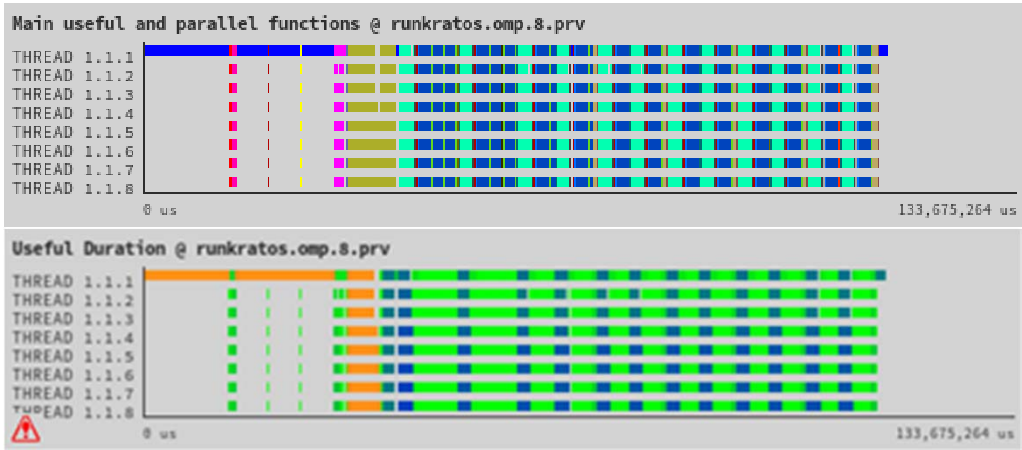






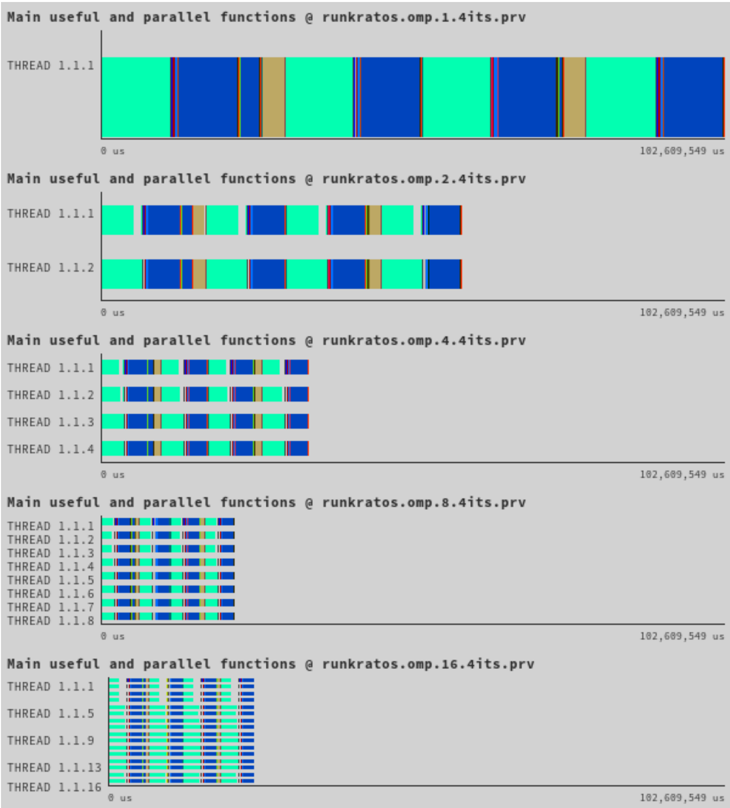
Structure

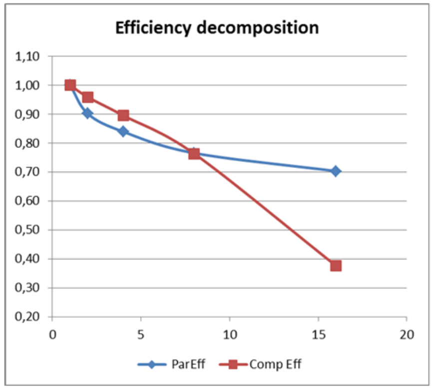


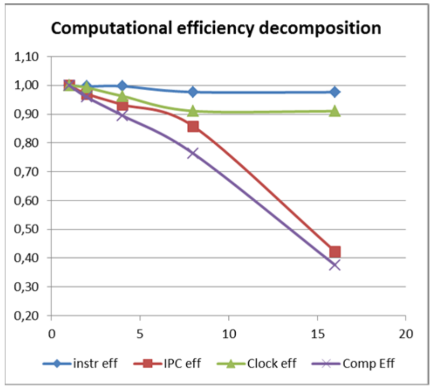


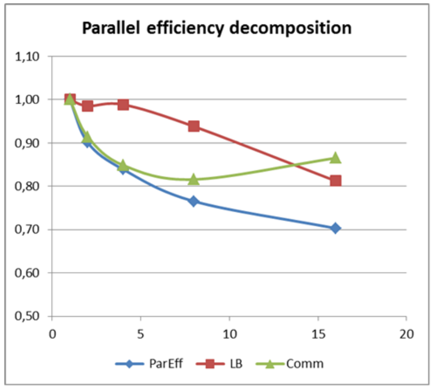
- In reality two different codes, similar structure
  - Multigrid non linear solver

OpenMP runs Scaling



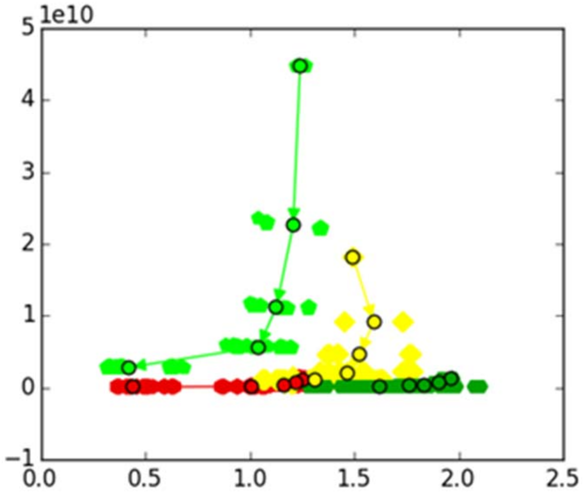






OpenMP runs efficiencies

- Serial performance (4 longest regions)



- Instruction efficiency: slight increase in total instruction count
  - Atomics ??

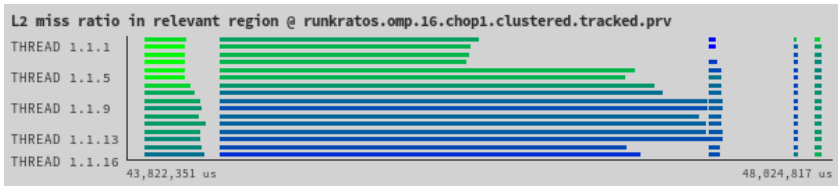
OpenMP Serial performance

- Longer 4 regions

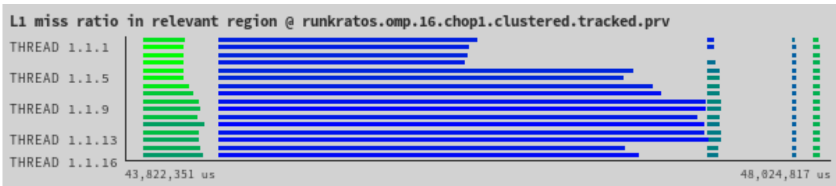
- Reason?
  - Computational?
  - NUMAness?
  - Numbering?
  - Combined?
  - None?

Instructions

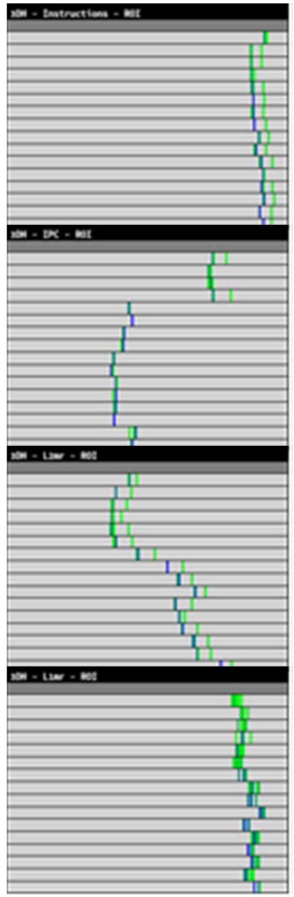
IPC



L2 miss ratio



L1 miss ratio

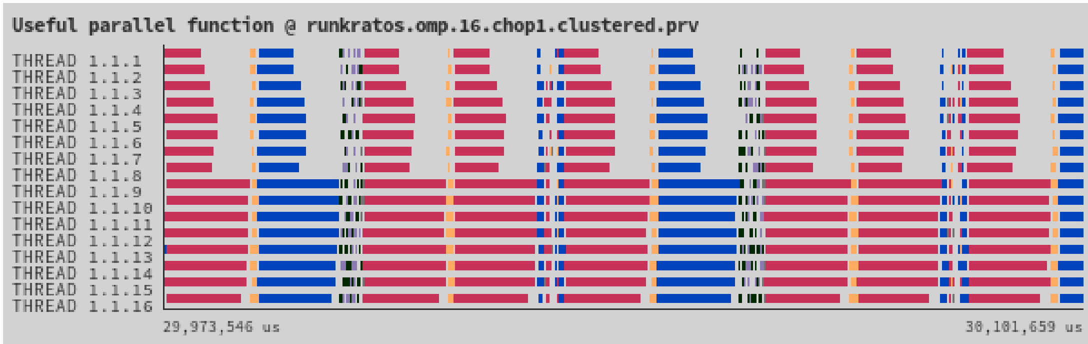


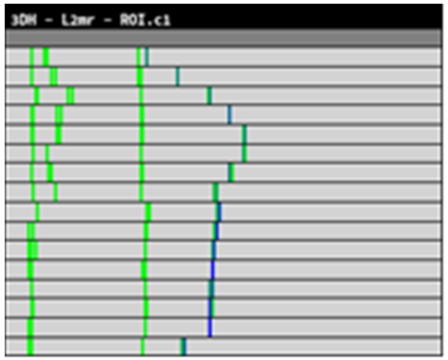
OpenMP Serial performance

- Finer grain regions

- Reason?
  - Computational?
  - NUMAness?
  - Numbering?
  - Combined?
  - None?

Outlined OpenMP functions





L2 Miss ratio

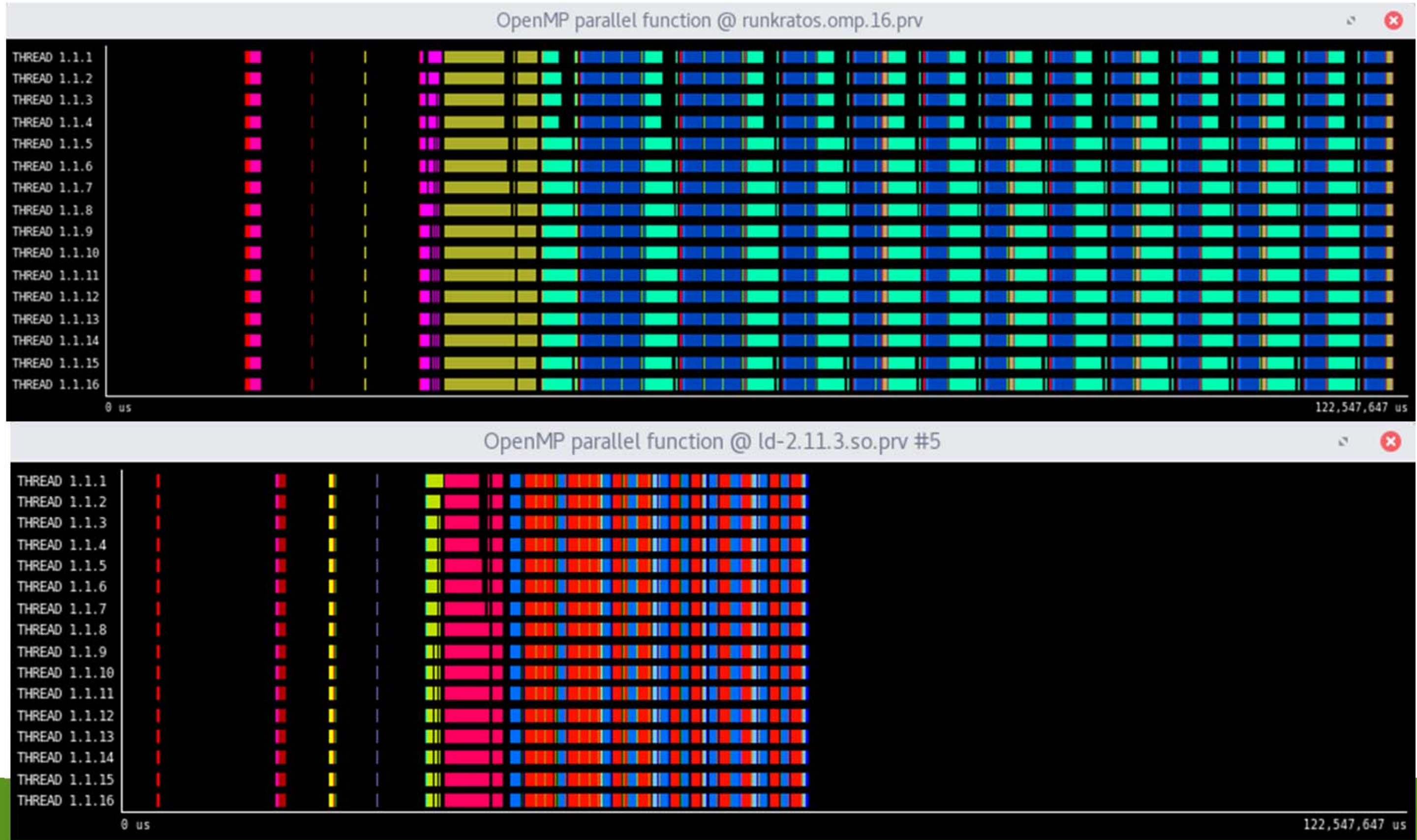
OpenMP assessment

- Many coupled effects
  - NUMAness, variability in cache miss ratios, atomics overheads (and contention?)

- Recommendations
  - NUMA initialization
    - Though they were doing it. Inadvertedly happened to be in the wrong control flow branch
    - Really activated → std::vector NUMA unfriendly issues …. Took some more time to fix
  - Explore potential benefits of more dynamic schedules
  - Work on numbering schemes
    - WIP: Not only balances IPC but also improves it
  - Eliminate atomics . Commutative multideps clause (OmpSs) ?
    - Verified high atomics overhead (running version with races)

Ongoing progress

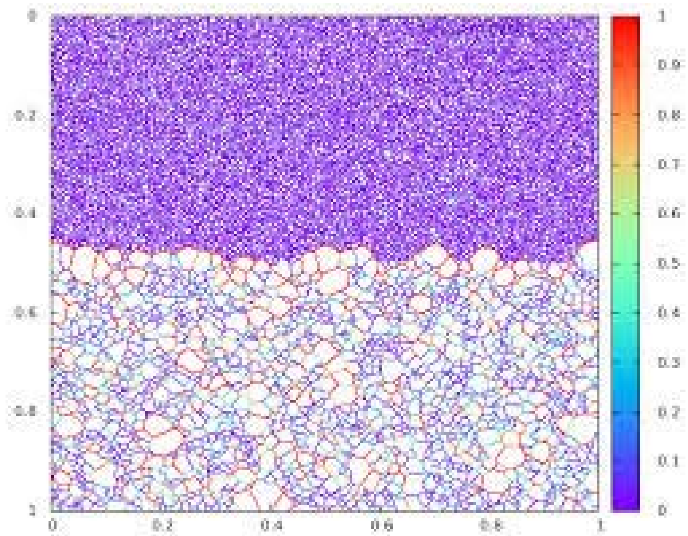
- Refactoring being implemented by customer



Ongoing progress
> 2x

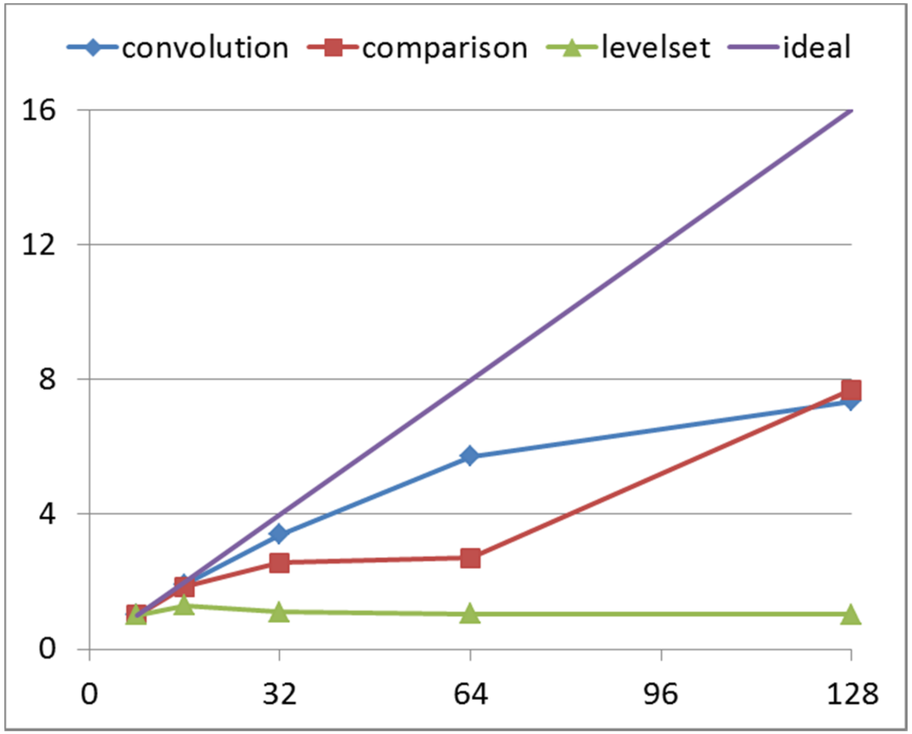
Case study: GraGLeS2D Audit

- User:
  - University
  - Developper
- Code: GraGLeS2D
  - Simulates the grain growth in polycrystalline materials
- Code Area: Material Science
- Performance Audit:
  - Poor scaling on a NUMA machine with 128 cores



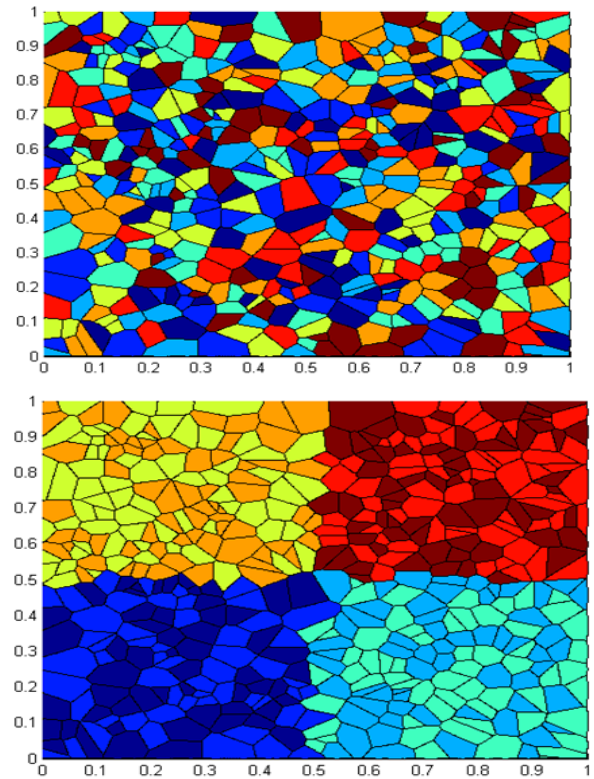
GraGLeS2D Audit Analysis

- Analysis of OpenMP with 8 – 128 cores
  - 4 boards x 4 sockets x 8 cores
- Observations from Audit
  - Work balance good except for the first iteration
  - Data sharing causing remote memory access reduces scalability
  - Detected consuming loops that can be vectorised
- PoC proposed and implemented



GraGLeS2D Proof of Concept

- PoC Plan
  - improve data-locality by thread pinning and load-distribution
  - improve vectorisation and serial performance



- Results on test input
  - parallel regions: speedup 6.4
  - overall application: speedup 2.2

Codes analyzed

- DPM
- Quantum Espresso
- DROPS
- Ateles
- SHP-Fluids
- GraGLeS2D
- NEMO
- VAMPIRE
- psOpen
- GYSELA
- AIMS
- OpenNN
- FDS
- Baleen
- Mdynamix
- ParFlow
- GITM
- BPMF
- FIRST
- SHEMAT
- GS2
- ADF
- DFTB
- ICON
- dwarf2-ellipticsolver
- EPW
- Code Saturne
- ONETEP
- Ms2
- SIESTA
- Oasys GSA
- SOWFA
- BAND
- NGA
- Fidimag
- LAMMPS
- ScalFMM
- CHAPSIM K.W.
- ArgoDSM
- CIAO
- FFEA
- k-Wave
- DSHplus
- RICH
- COOLFluiD
- Ondes3D
- ATK
- Molcas
- GBMol_DD
- Kratos
- cf-python

+ few under NDAs

Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.