How to Improve the Performance of Parallel Codes

Jon Gibson, NAG

EU H2020 Centre of Excellence (CoE)

1 October 2015 – 31 March 2018

Grant Agreement No 676553

Contents

- Code Performance
- Profiling and Optimisation
- Live Demo
- Real Examples of Code Improvements

The POP Service

- **P**erformance **O**ptimisation and **P**roductivity
- A Centre of Excellence
  - Collaborative European project funded by Horizon 2020 programme
  - Runs October 2015 – March 2018
- Providing Free Services within Europe
  - Precise understanding of **parallel** application and system behaviour
  - Across application areas, platforms and scales
  - Suggestions/support on how to rewrite code in the most productive way
  - For academic and industrial codes and users

The POP Team

- Participating institutions:
  - Barcelona Supercomputing Center, Spain (coordinator)
  - HLRS, Germany
  - Jülich Supercomputing Center, Germany
  - NAG, UK
  - RWTH Aachen, IT Center, Germany
  - TERATEC, France
- A team with:
  - Expertise in performance analysis and optimisation
  - Expertise in parallel programming models and practices
  - A research and development background and a proven commitment to real academic and industrial use cases

Why improve performance?

- Time is money – especially on supercomputers
- To run bigger and/or more complex simulations
- To remain competitive with similar codes

Understanding Performance is Hard

- Scientific Codes
  - Often large codes developed by many people
  - Development driven by functionality rather than performance
  - Difficult to get an overview of the code's behaviour
- HPC machines
  - Complex architectures
    - Many nodes, each consisting of a number of multicore processors
    - An interconnect and a filesystem
    - Vector operations
    - Deep memory hierarchies with a number of levels of cache
  - Not easy to program efficiently

Where do we start?

- Are there any easy wins?
  - Are we using the best performing compiler for our code?
  - With the best choice of compiler flags?
  - And the best performing MPI library?
- We need to be very selective before spending time optimising code
  - *"Premature optimization is the root of all evil."* – Donald Knuth
  - Optimising code is often time-consuming
  - Optimised code is often more difficult to read/understand (hence debug/maintain)
  - Optimising a routine that only takes 2% of the execution time is going to have very little impact on the overall performance
- We therefore need a way to understand the behaviour of a code in order to guide the optimisation process

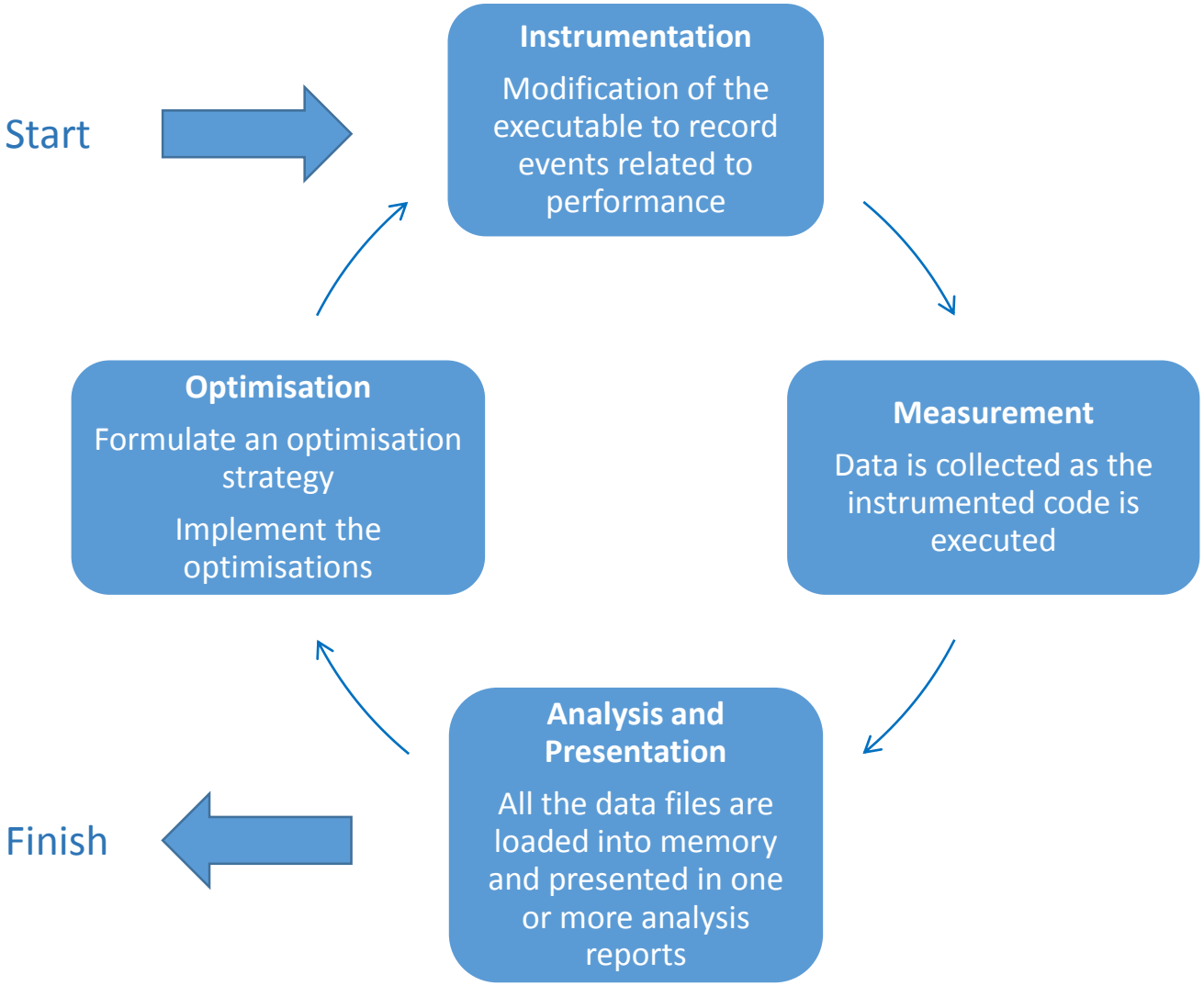
Profiling and Performance Analysis

- *Profiling* refers to the monitoring of a code's behaviour as it executes
- There are a number of profiling tools available, which by helping to answer a number of key questions, allow us to optimise effectively
  - What are the most time-consuming routines?
  - What are the most time-consuming lines in those routines?
  - Is it easy to optimise or is the efficiency already high?
  - What needs to be optimised, i.e. what is the bottleneck?
    - Cache efficiency, vectorisation, etc
  - For a parallel code, is it load-balanced?
    - Essential if the code is to scale
  - How many MPI messages are there and what size are they?

Some Profiling Tools

- Gprof – GNU Profiler
- PAPI – Performance Application Programming Interface
- TAU – Tuning and Analysis Utilities
- Scalasca
- Extrae and Paraver
- Allinea MAP
- HPCToolkit
- OpenSpeedShop
- Vampirtrace and Vampir
- …and many others.

The Profiling-Optimisation Cycle



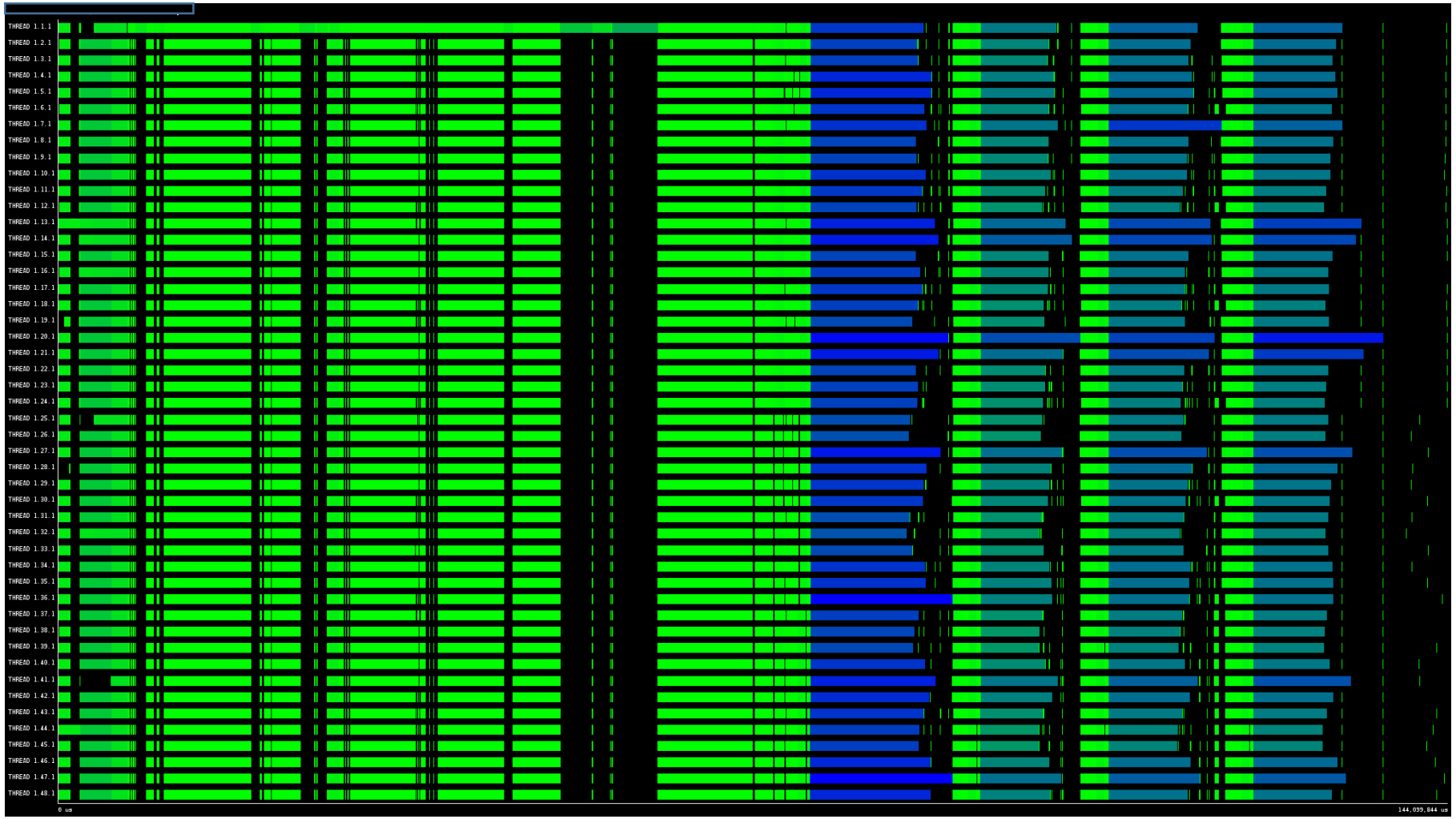
The Details of the Run

- The Input Data
  - Profiling results, and therefore possible bottlenecks, are likely to change with different input files.
  - Ideally, therefore, we want to profile a typical production run rather than a trivial test case.
- The Number of Cores
  - Profiling results are likely to change when the job is run on different numbers of cores.
  - When a code does not scale well, profiling it on different numbers of cores will help identify the cause of the poor scaling.
  - Ideally, profile on the number of cores you aim to scale up to.

A Real Example

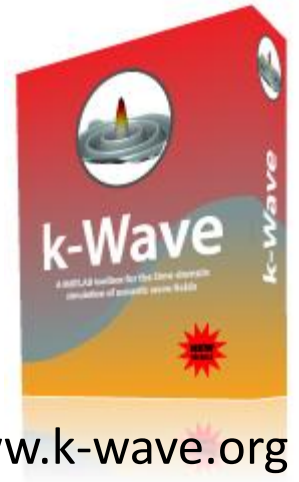
- As part of a current investigation, data has been collected for this code being run on ARCHER using the tool *Extrae*.
- We will take a look at how this data can be visualised using *Paraver*.
  - Extrae and Paraver are developed by Barcelona Supercomputing Center

48 MPI processes for 4 time steps



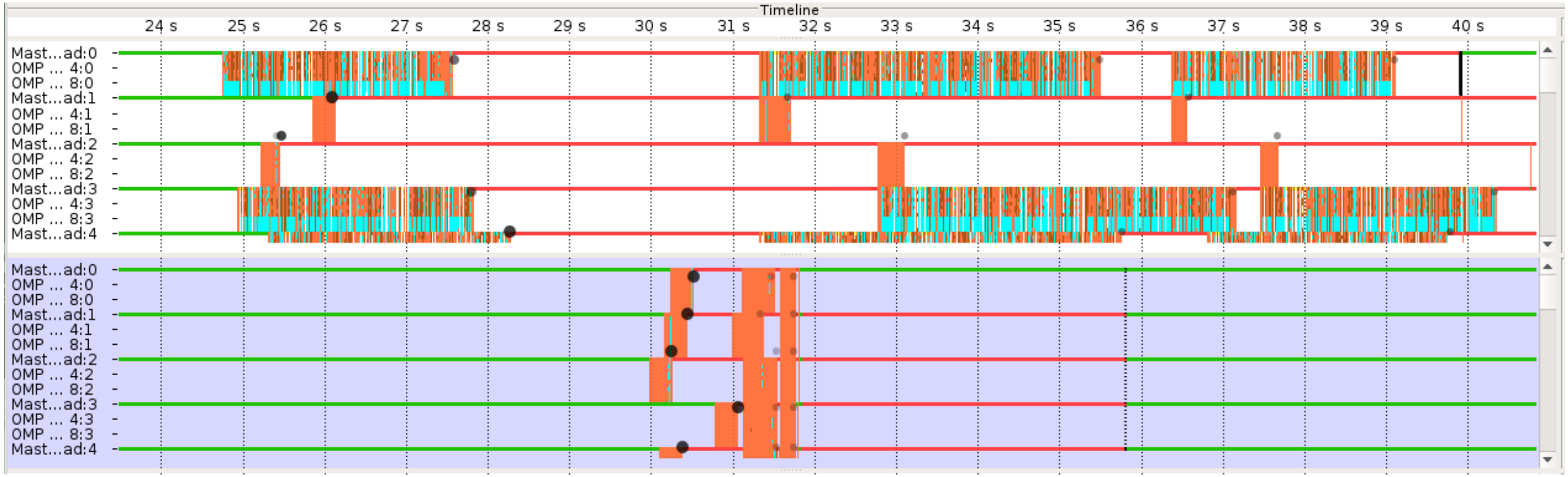
k-Wave – Brno Uni. of Technology

- Toolbox for time domain acoustic and ultrasound simulations in complex and tissue-realistic media
- C++ code parallelised with Hybrid MPI and OpenMP (+ CUDA)
- Profiling showed that
  - 3D domain decomposition suffered from major load imbalance: exterior MPI processes with fewer grid cells took much longer than interior
  - OpenMP-parallelised FFTs were much less efficient for grid sizes of exterior, requiring many more small and poorly-balanced parallel loops
- Using a periodic domain with identical halo zones for each MPI rank reduced overall runtime by a factor of 2



www.k-wave.org

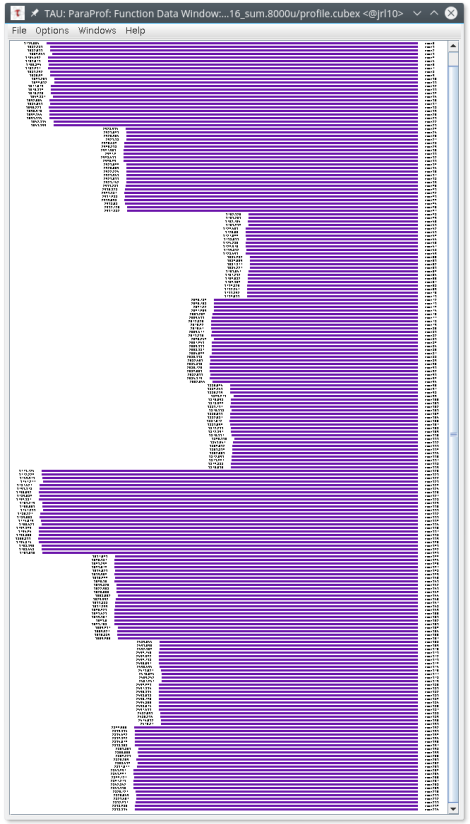
k-Wave – Brno Uni. of Technology



- Comparison time-line before (top) and after (bottom) balancing, showing exterior MPI ranks (0,3) and interior MPI ranks (1,2)
  - MPI synchronization in red; OpenMP synchronization in cyan

EPW – University of Oxford

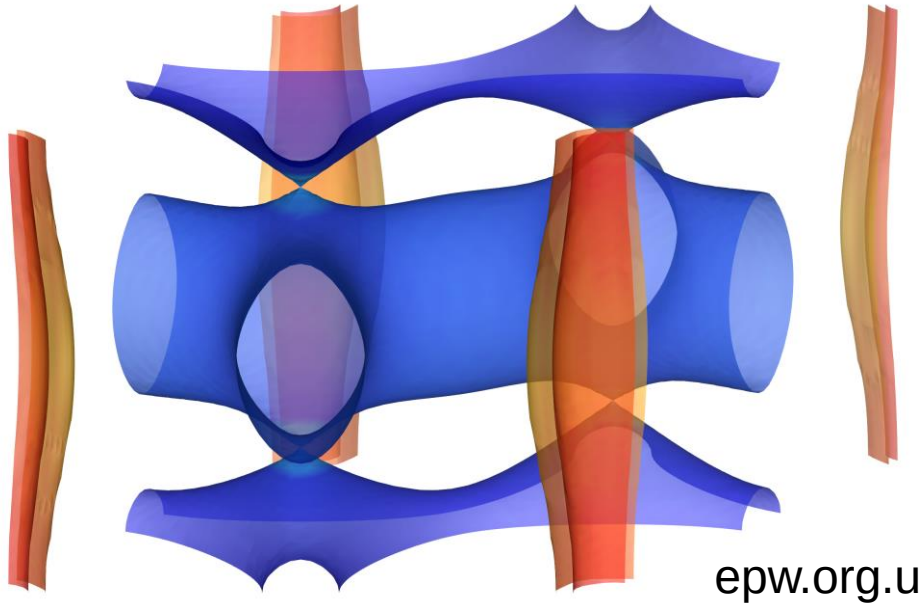
- Electron-Phonon Wannier (EPW) materials science DFT code; part of the Quantum ESPRESSO suite
- Fortran code parallelised with MPI
- Profiling showed
  - Poor load balance
  - Large variations in runtime, likely caused by I/O
  - Final stage spends a great deal of time writing output to disk



EPW – University of Oxford

- Original code had all MPI processes writing result to disk at the end
- This was modified this so that only one rank wrote the output
- On 480 MPI processes, time taken to write results fell from over 7 hours to just 56 seconds: a 450-fold speed-up!

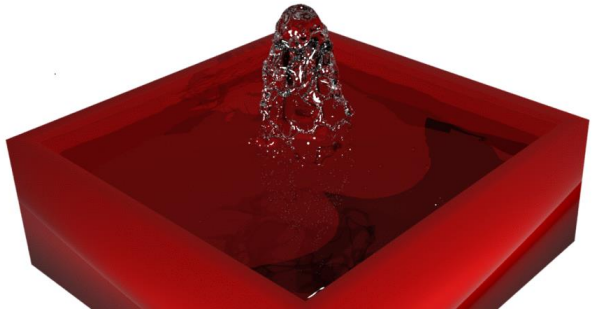
- Combined with other improvements, this enabled simulations to scale to a previously impractical 1920 MPI processes



epw.org.uk

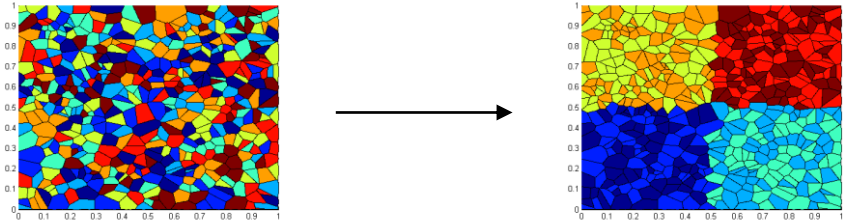
sphFluids – Stuttgart Media University

- Smoothed particle hydrodynamics code
  - C++ with OpenMP
- Profiling identified several issues
  - Definitions of variables in inner loops
  - Unnecessary operations caused by indirection in code design
  - Frequently-used non-inlined functions
  - High cache misses, which could be reduced by reordering the processing of particles
- The developers decided to completely rewrite the code based on their new knowledge, leading to an overall performance improvement of 5x - 6x



GraGLeS2D – RWTH Aachen

- Simulation of microstructure evolution in polycrystalline materials
- After profiling, the following optimisations were implemented
  - Memory allocation library optimised for multi-threading
  - Reordering the work distribution to threads



  - Algorithmic optimisation in the convolution calculation
  - Code restructuring to enable vectorisation
- An improvement of over 10x was demonstrated for the region concerned, with an overall application speed-up of 2.5x

What does POP do?

? Performance Audit ⇨ Report
- Identify performance issues of customer code
- Small effort (< 1 month)

! Performance Plan ⇨ Report
- Follow-up on the audit service
- Identifies the root causes of issues and qualifies/quantifies fixes
- Longer effort (1-3 months)

✓ Proof-of-Concept ⇨ Software Demonstrator
- Experiments and mock-up tests for customer codes
- Kernel extraction, parallelisation, mini-apps, …
- 6 months effort

Who are POP targeting?

- Code Developers
  - Assessment of detailed behaviour of code
  - Suggestion of most productive directions to refactor code
- Users & Infrastructure Operators
  - Assessment of achieved performance in production conditions
  - Possible improvements from modifying environment setup
  - Evidence to interact with code provider
  - Training of support staff
- Vendors
  - Benchmarking, customer support and system design

Further Information

- A selection of training materials is available on the POP website
  - https://pop-coe.eu/further-information/learning-material
- Please direct any questions regarding this webinar to pop@nag.co.uk
- and if you're interested in using the POP service, then e-mail pop@bsc.es
- We hope to see you at future webinars!
  - Please let us know if there's anything that you'd particularly like to hear about.

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.