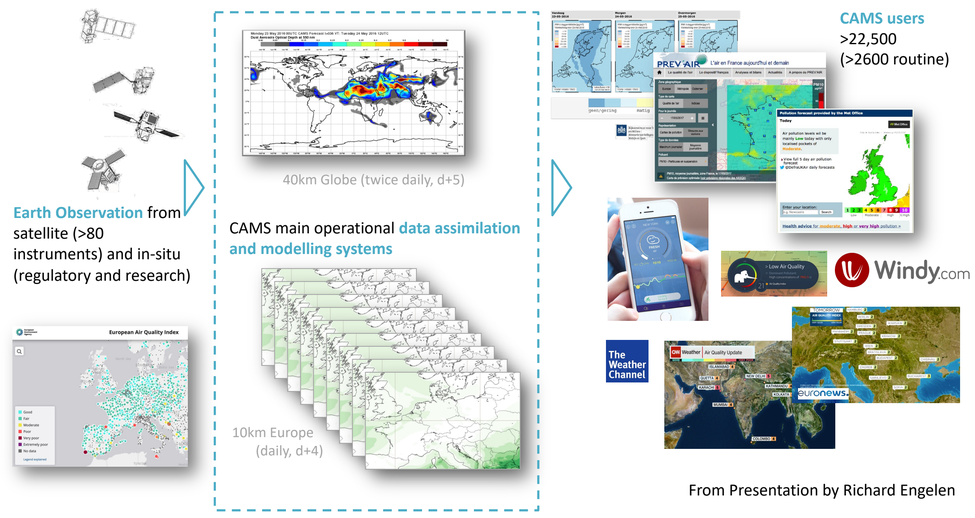
Copernicus Climate Change Service
Copernicus Atmosphere Monitoring Service

Introduction to CAMS Mini-Application

Iain Miller*, Olivier Marsden^ and Lucian Anton* – ECMWF, *Computing Department, ^Research Department

The Copernicus Atmosphere Monitoring Service, CAMS, is an integrated part of the IFS code, which is used to provide consistent and quality-controlled information related to air pollution and health, solar energy, greenhouse gases and climate forcing everywhere in the world.
Users then can use this information to implement mitigation measures in a timely manner.



From Presentation by Richard Engelen

What does the code do?

The bulk of the calculation is in a generated section that solves a stiff ODE system for each chemical species. For this it uses an implicit Rosenbrock ODE solver[1,2] that is efficient for solving stiff systems like this.

Limitations

CAMS tracks a lot more chemical species and fields than a regular IFS forecast, which along with the usage of the additional solvers, means that runtimes when CAMS is used are 6-8 times that compared to not having it enabled. Therefore, we want to be able to assess the cost of different configurations, solvers and methods in isolation from the IFS costs.

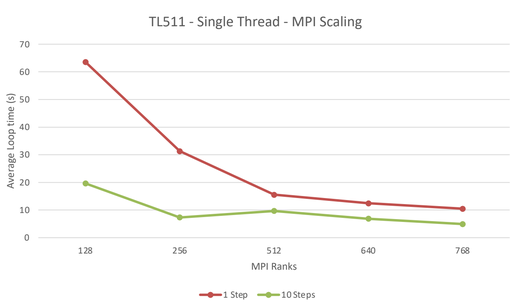
Developing the Mini-Application

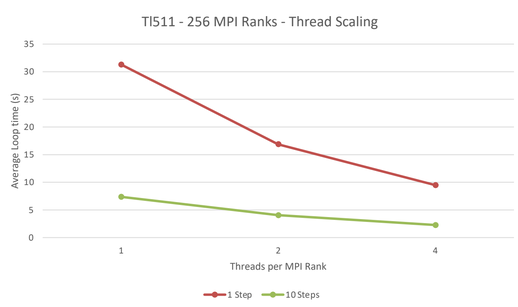
The mini-app has been developed as a subset program of the IFS code.

- The input and validation data is recorded during a standard CAMS-IFS forecast run, either side of the the first call to `chem_main` (the CAMS entry point).
- The mini-application then wraps the call to `chem_main` with the input data structures setup by the standard IFS setup routines and populated from the inputs.
- In this way we can keep any changes close to the current IFS code without being directly affected by them and then change the CAMS code in isolation and assess the impact of those changes.
- Also allows us to profile the CAMS code sections and find the hotspots without interference from any IFS routines.

Initial Findings

- Scaling is linear in terms of both MPI ranks and threads per rank.
- Unsurprisingly, the bulk of the time is spent in the Rosenbrock routines.
- Most expensive lines are in areas where the loop order is not conducive to contiguous memory access and effects vectorisation.





Current Progress

- Re-ordering loops and promoting subroutines and variables in highest cost loops.
- Removed extraneous Dr Hook timing statements.
- Fixed some memory leaks.

References and Acknowledgements

[1] Valeriu Damian, Adrian Sandu, Mirela Damian, Florian Potra, Gregory R. Carmichael, The kinetic preprocessor KPP-a software environment for solving chemical kinetics, Computers & Chemical Engineering, Volume 26, Issue 11, 2002, Pages 1567-1579, ISSN 0098-1354, https://doi.org/10.1016/S0098-1354(02)00128-X

[2] KineticPreProcessor, KPP, Github repository, https://github.com/KineticPreProcessor/KPP

The authors would like to acknowledge the help, time and support in this project from Johannes Flemming, Mihai Alexe, Vincent Huijnen, Zak Kipling and Ioan Hadade.

PROGRAMME OF THE EUROPEAN UNION
IMPLEMENTED BY ECMWF