HPC Software Optimisation Highlights from the POP Project

Jonathan Boyle   Sally Bridgwater   Nick Dingle   Jon Gibson   (Numerical Algorithms Group)

Background

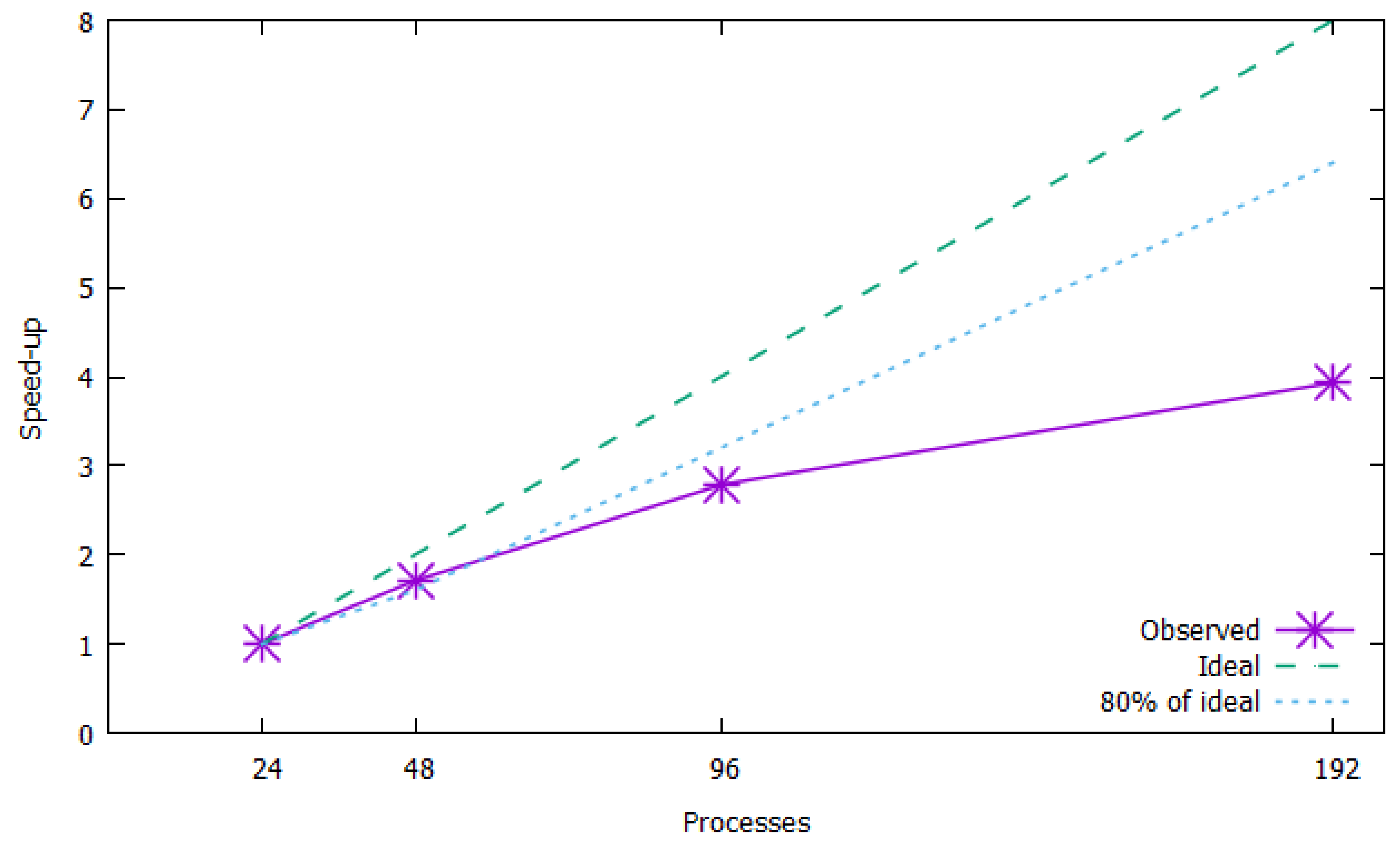
As part of the EU-funded Performance Optimisation and Productivity (POP) Centre of Excellence, NAG offers HPC expertise to organisations who want help to understand and improve the performance of parallel software.

This poster describes NAG's work on three different codes: WRF-Chem, ADF and zCFD.

WRF-Chem

The Weather Research and Forecasting (WRF) Model is a mesoscale numerical weather prediction system designed for both atmospheric research and operational forecasting needs.

WRF-Chem couples chemistry into this model, enabling it to simulate the emission, transport, mixing, and chemical transformation of trace gases and aerosols simultaneously with the meteorology. WRF-Chem is used for investigation of regional-scale air quality, field program analysis, and cloud-scale interactions between clouds and chemistry.



For the test case used we observed little speed-up in going from 96 to 192 cores. The main reason for this was poor computational load balance between the MPI ranks. In addition a lot of time was spent in `MPI_Wait` waiting for `MPI_Irecv` calls to complete.

A further study is now underway to investigate how the load balance varies across the geographical decomposition in order to find ways in which it might be improved.

ADF

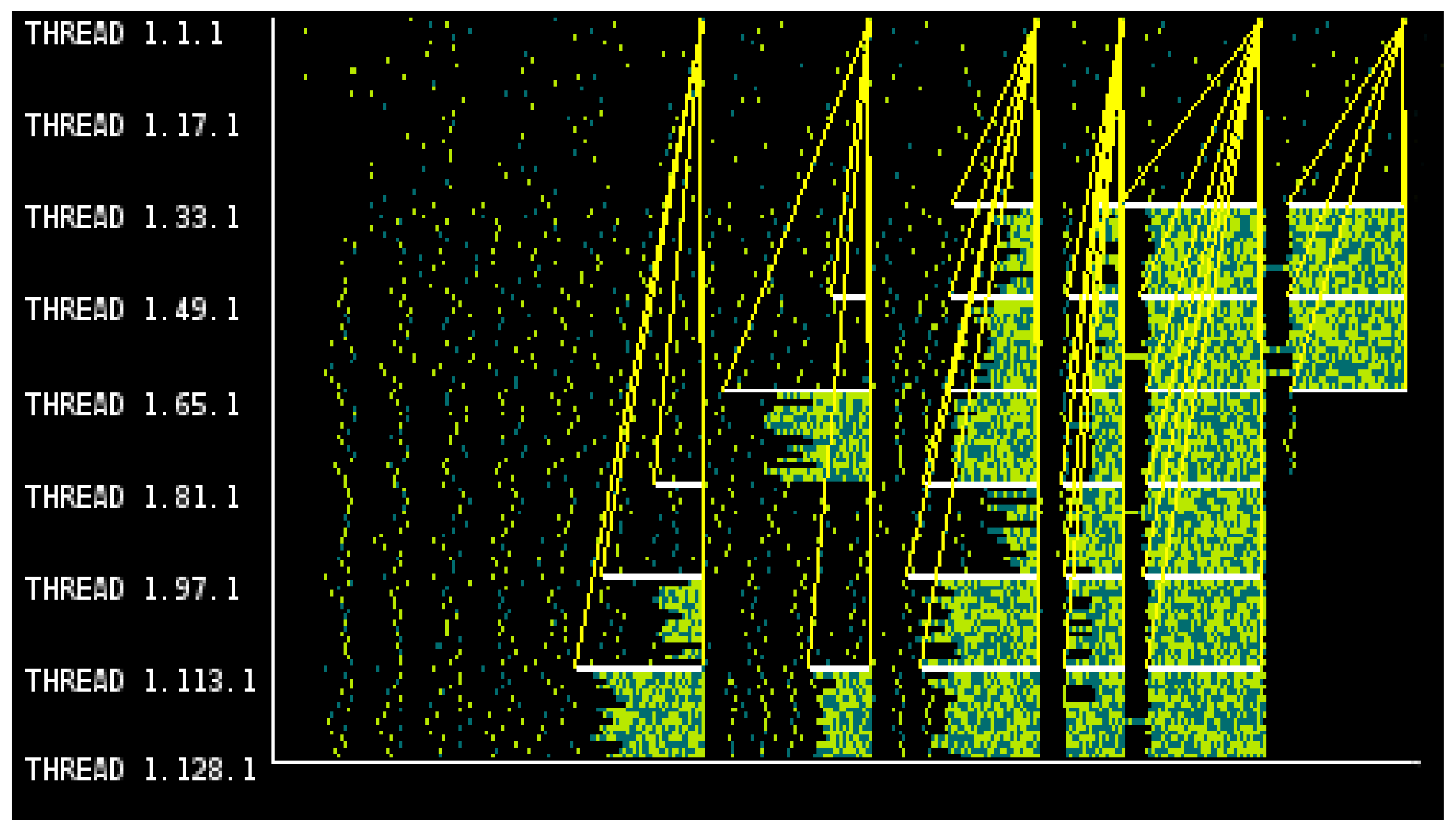
ADF is the flagship code from the Netherlands-based company Software for Chemistry and Materials (SCM). It is a computational chemistry application which uses density functional theory calculations to predict the structure and reactivity of molecules.

NAG assessed the performance of a particular calculation (medium-sized molecule with hybrid exchange-correlation functional) and identified that the main issue was computational load imbalance:

Code Region 1	16	32	64	128
Global Efficiency	0.98	0.84	0.61	0.52
Computational Scalability	1.00	0.94	0.86	0.76
Parallel Efficiency	0.98	0.89	0.71	0.68
Load Balance	1.00	0.93	0.76	0.77
Communication Efficiency	0.99	0.96	0.94	0.89
Serialization Efficiency	0.99	0.99	0.98	0.95
Transfer Efficiency	0.99	0.97	0.96	0.93
IPC Scalability	1.00	0.99	0.96	0.93
Instructions Scalability	1.00	0.98	0.95	0.91
Frequency Scalability	1.00	0.96	0.93	0.89

Code Region 2	16	32	64	128
Global Efficiency	0.99	0.76	0.46	0.39
Computational Scalability	1.00	0.94	0.83	0.72
Parallel Efficiency	0.99	0.81	0.56	0.54
Load Balance	0.99	0.83	0.58	0.60
Communication Efficiency	1.00	0.98	0.96	0.90
Serialization Efficiency	1.00	0.99	0.99	0.97
Transfer Efficiency	1.00	0.98	0.97	0.93
IPC Scalability	1.00	0.99	0.96	0.95
Instructions Scalability	1.00	0.99	0.98	0.96
Frequency Scalability	1.00	0.94	0.82	0.74

We identified that this was because node masters can spend a lot of time waiting to receive work from the global master rank, which leads to a significant amount of idle time for the rest of the ranks in a node.



The horizontal white lines show the times that the node masters are waiting to get more data from the global master. The yellow lines connect the start and end points of communications. The green and blue points are the slave ranks checking to see if new work is available.

We estimated that improving the load balance could lead to a 2x increase in performance. SCM implemented a dynamic load balancing scheme and observed a reduction in runtime from 4.2s to 2.0s on a particular calculation.

zCFD

zCFD by Zenotech (`https://zcfd.zenotech.com/`) is a density based finite volume and Discontinuous Galerkin (DG) computational fluid dynamics (CFD) solver for steady-state or time-dependent flow simulation. It decomposes domains using unstructured meshes.

zCFD comprises a Python package (`zCFD-driver`) that calls computational kernels written in C++. NAG investigated the performance of the C++ part of the DG solver only.

Based on our findings Zenotech made a number of changes to the code:

- Changing the `slurm` CPU governor setting to boost performance on heavily loaded cores.
- Removing an `inline` keyword allowed the compiler to better optimise the code and ensured all OpenMP pragmas were enabled.
- Adding a small extra calculation avoided going through a very slow codepath in `pow()` and thus incurring additional load imbalance.
- Switching from `static` to `dynamic` OpenMP scheduling with dynamically adjusted choice of chunksizes.
- Increasing average CPU utilisation by not creating OpenMP regions on multiple threads.
- Tweaking memory management, e.g. when calling BLAS routines.

These led to a 3x performance improvement on 12 OpenMP threads.

Acknowledgements

We would like to thank the University of Manchester (WRF-Chem), SCM (ADF) and Zenotech (zCFD) for giving us permission to showcase the work they have undertaken with POP.

The other POP project partners are Barcelona Supercomputing Center, Juelich Supercomputing Center, RWTH Aachen, HLRS Stuttgart and TERATEC. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 676553.