Analysis and Improvement of HPC Software Performance: ADF Modeling Suite

Jonathan Boyle *Sally Bridgwater* *Nick Dingle* *(Numerical Algorithms Group)*

Background

As part of the EU funded *Performance Optimisation and Productivity* (POP) Centre of Excellence, NAG is offering HPC expertise to organizations who want help to understand and improve poor performance of their parallel codes.

This poster describes POP work to improve multiplication of small complex matrices in SCM's ADF Modeling Suite, within periodic DFT (density functional theory) computation for nanotubes, surfaces, and bulk material.

Parallel complex matrix multiplication

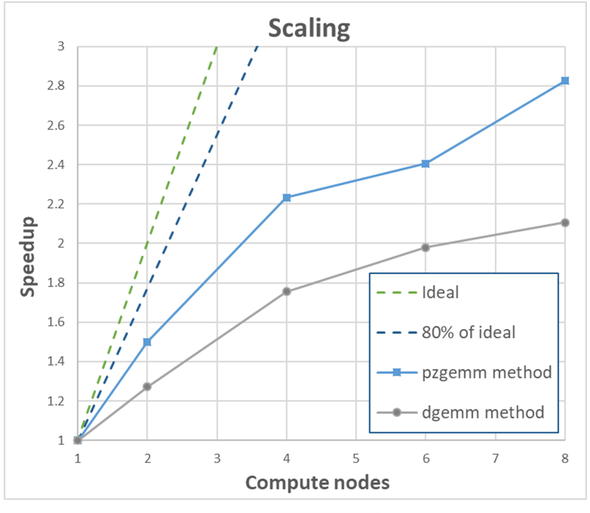
The software is written in Fortran and MPI, and each matrix is stored in two POSIX shared memory real arrays, which hold the real and imaginary components of the matrix. Each matrix is duplicated on every compute node, and two multiplication methods have been implemented:

1. Each MPI process calls BLAS dgemm 4 times to compute a real and imaginary block of the result matrix.
2. Shared memory matrices are converted to distributed matrices, and PBLAS pzgemm computes the result matrix.

For both methods MPI collective communications are used to convert the result data to a shared memory array on each node.

Parallel scaling

The graph shows the poor parallel scaling seen for small matrices, for this case matrices are 3200 × 3200. Each compute node has 16 cores.



Performance analysis

We analysed parallel performance using a range of metrics which are designed to expose the specific causes of poor scaling, for example:

- Transfer Efficiency: loss of efficiency due to time in MPI data communication
- Serialization Efficiency: loss of efficiency due to wait time within MPI
- Computational, IPC & Instruction Scalability: to identify if total useful computational work is increasing.

The full set of metrics are described on the POP CoE website.

This table shows a subset of performance metrics for matrices of size 3200 × 3200, on 1, 4 and 8 compute nodes of the Barcelona Supercomputing Center MareNostrum III hardware. Green indicates good efficiency/scaling values, and red indicates poor values.

	pzgemm method			dgemm method		
Number of nodes	1	4	8	1	4	8
Global Efficiency	0.74	0.41	0.26	0.95	0.42	0.25
Load Balance Efficiency	0.93	0.81	0.74	0.98	0.95	0.89
Serialization Efficiency	0.99	0.97	0.91	0.97	0.93	0.89
Transfer Efficiency	0.81	0.56	0.46	1.00	0.74	0.67
Computational Scalability	1.00	0.94	0.86	1.00	0.64	0.47
IPC Scalability	1.00	0.98	0.93	1.00	0.75	0.64
Instructions Scalability	1.00	0.99	0.97	1.00	0.90	0.78

Performance bottlenecks for pzgemm method

For the pzgemm method, the important performance bottlenecks are:

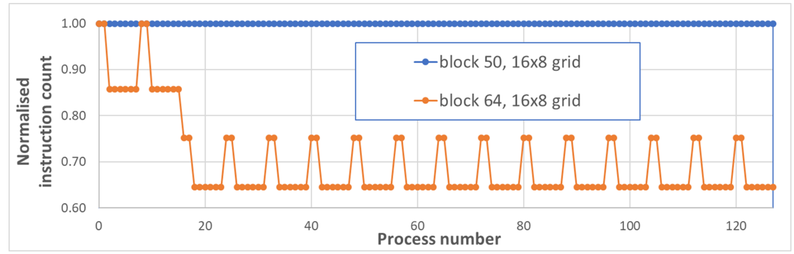
1. Very low transfer efficiency
2. Low load balance efficiency.

The MPI data transfer is largely unavoidable for this method. The load imbalance was analysed and found to be caused by an imbalance in useful instructions within pzgemm, with a clear relationship between the number of instructions per process and the distribution of matrix blocks on the BLACS process grid.

By modifying the block distribution, we were able to remove the load imbalance (see graph in next column). However, this improved load balance was at the expense of reduced transfer efficiency and serialization efficiency, with no overall reduction in run time.

Instructions distribution for pzgemm method

This data shows the number of useful instructions per process for matrix block sizes of 50 and 64, the load imbalance is removed for the block size of 50.



Performance bottlenecks for dgemm method

The low parallel scaling for the dgemm method is caused by:

1. Low transfer efficiency
2. Very low computational scalability, caused by increasing useful instructions and reducing IPC.

Our analysis of the algorithm identified a strategy to remove half the collective communications needed to share the result matrix over the compute nodes.

The increasing computational work occurs within dgemm, and our work has identified more efficient block strategies for this method. This is estimated to raise computational scalability from the current value of 0.47 up to 0.91, on eight compute nodes.

Overall the improvements to MPI communications and computational scaling are estimated to give a 2x speed up for the 3200 × 3200 matrix multiplication. These ideas are currently being tested as part of a POP Proof of Concept study.

Acknowledgements

POP is funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.

We'd like to give grateful thanks to SCM (`www.scm.com`) for allowing us to present these results.