Proceedings of the 24th UK Conference of the
Association for Computational Mechanics in Engineering
31 March – 01 April 2016, Cardiff University, Cardiff

Development of a High Performance Computing Approach for Studying the Coupled Behaviour of Porous Media

***Manhui Wang [1], Lee Hosking [1], Shakil Masum [1] and Hywel Thomas [1]**

[1]Geoenvironmental Research Centre, School of Engineering, Cardiff University,
Queen's Buildings, The Parade, Cardiff CF24 3AA, United Kingdom

*wangm9@cardiff.ac.uk

ABSTRACT

High-performance computing (HPC) techniques, especially parallel computing, can be used to significantly reduce the computational time in very large scale simulations. In this work, the popular HPC hardware and parallel programming tools are reviewed and discussed. Then a hybrid Message Passing Interface/Open Multi-Processing (ie, MPI/OpenMP) model is proposed to parallelise the numerical code COMPASS, which adopts a coupled thermal/hydraulic/chemical/mechanical (THCM) approach to model the behaviour of saturated/unsaturated media. Various optimisation techniques have also been employed to achieve better performance. Together with code optimisation, the parallel COMPASS has shown significant performance improvement, which has been seen for an example application in carbon dioxide sequestration.

Keywords: *high performance computing; parallel computing; THCM; COMPASS*

1. Introduction

Geoenvironmental problems, such as contaminated land, landfills, waste disposal, and carbon dioxide sequestration, are generally based upon flow conditions through soils and/or chemical reactions. For example, the flow of carbon dioxide through rocks/soils and the associated chemical processes are important factors controlling the capacity and long term stability in carbon dioxide sequestration. These problems are often complex in respect to both the processes occurring and their geometry and scale. In recent years, a coupled thermal/hydraulic/chemical/mechanical (THCM) approach to model the behaviour of saturated/unsaturated media has been developed at the Geoenviromental Reseach Cenre (GRC) at Cardiff University [1-4]. The model has been implemented in the numerical computer code COMPASS (COde for Modelling PArtially Saturated Soils), and used to simulate a number of problems. The finite element method is employed for the spatial discretisation, and a finite difference time stepping scheme is implemented for temporal discretisation.

As the scale and complexity in geoenergy applications increases, the demands for computational resources increase dramatically and eventually become time-limiting for very large simulation problems. Recent developments of high-performance computing (HPC) provide great advantages to overcome these large computational demands and time-limiting issues. Parallel computing on HPC platforms can significantly reduce the computational time. It is worthwhile to invest time and efforts to develop numerical modelling code that run on a HPC system. In this paper, the authors briefly demonstrate the development of parallel version of the GRC's in-house numerical modelling code, COMPASS, with cutting-edge HPC technologies. It aims to enable the integrated parallel COMPASS code to run on supercomputers efficiently, reliably and quickly.

2. HPC hardware and parallel programming tools

The November 2015 Top 500 ranking of world supercomputers [5] shows that more than 80% of HPC machines are classified as having a cluster architecture. HPC clusters usually consist of a large number of distributed computer nodes linked through high performance networks. A well-known transport mechanism between nodes is message passing, which was adopted by many parallel

programming tools, such as Theoretical Chemistry Group Message Passing System (TCGMSG) [6], Parallel Virtual Machine (PVM) [7], and Message Passing Interface (MPI) [8]. With a number of features that provide both convenience and high performance communication, MPI is the most widely used model, and has become the de facto standard for incorporating parallelism into scientific and other application codes.

Table 1: Comparison of features between MPI and OpenMP

MPI	OpenMP
Shared or distributed memory architectures	Only shared memory architecture
Requires an MPI library(MPICH, Intel MPI etc)	Doesn't need other library
No specific requirements for compiler	Requires a compiler that supports OpenMP
Message based	Directive based
Both process and thread based approach	Only thread based parallelism
Overhead for creating process is one time	Threads can be created and joined for particular task which add overhead
There are overheads associated with transferring message between processes	No such overheads, as thread can share variables
Process in MPI has private variable only, no shared variable	Thread can have both private and shared variables
Requires more programming changes to go from serial to parallel version	Directives can be added incrementally, less changes from serial to parallel version
Rich functionality, flexible, harder to program	Easier to program, data racing bug

In a HPC cluster, each computer node itself is a symmetric multiprocessor (SMP) system, on which memory is shared across the processors. Open Multi-Processing (OpenMP) [9] has provided a very rich and flexible programming model for such shared memory systems. Now most computers, even desktops/laptops, have multi-core CPUs, which can have access to the common memory on the machine. This computer architecture provides great convenience for parallelism, and OpenMP becomes one of the most popular programming models as it is easy to implement. Table 1 lists the major features of MPI and OpenMP. In order to achieve scalable performance, many application programs typically maintain and manipulate global data structures that are distributed between the memory managed by individual processors. The application programming interfaces (API) that provide such functionality include the Global Arrays (GA) toolkit [10], Distributed Data Interface (DDI) [11], and Parallel Programming Interface for Distributed Data (PPIDD) [12].

With the development of new accelerator technologies, general-purpose graphical processing units (GPGPUs) now can be used for parallel computing [13]. CUDA and OpenCL are most widely used GPU parallel programming tools. The potential performance improvement is enormous if the algorithms can take advantage of the GPU's programming model and do most of their computation on the GPU. There are cases of over a 100X performance improvement for some codes running on GPUs relative to CPUs. But one of the critical limitations is that you have to rewrite the code for the GPU. It is particularly hard to port existing complex code for GPU parallel computing.

Basing on the above analysis and discussions, it is a good option to develop parallel implementation with both MPI and OpenMP for currently existing large packages, in order to take full advantage of modern HPC facilities available.

3. Parallel strategies for COMPASS

As the popular HPC clusters have the hybrid architecture, where memory is distributed across the computer nodes but shared within the node, a hybrid MPI/OpenMP model is proposed for parallel implementation for the numerical code COMPASS. The hybrid model can take advantage of the merits of both MPI and OpenMP. MPI facilitates efficient inter-node reductions and transferring of complex data structures, and it helps the application scale across multiple SMP nodes in the HPC cluster. OpenMP manages the workload on each SMP node, and makes more efficient use of the shared memory. By spawning several OpenMP threads for each MPI process, less replicated data and less memory are required due to the decrease of the total number of MPI processes. Within each SMP node where only one MPI process is allocated, the synchronization is implicit, which eliminates much of the overhead associated with message passing.

In COMPASS, the majority of the computation time is spent in two sections: *system matrix build* and *solver*. The *system matrix build* process is to form the matrices for each element, and then assemble them into global matrix. In this section, coarse-grained domain decomposition can be employed as there is no communication in the matrix forming process for each element. Only at the end are all element matrices synchronised and added together. The *solver* is to provide the solution for the equations involved. In general, there are two types of solvers, direct and iterative [14]. A direct solver, such as LU decomposition, is very robust and efficient, but for large and complex systems, the memory required may become a bottleneck. Iterative solvers require lower memory, which is crucial for large scale simulations. Among these iterative solvers, the conjugate gradient algorithm is one of the most effective iterative methods.

For LU decomposition, fine-grained parallelism is achieved during the forward and back substitution. In conjugate gradient algorithms, vector-vector and matrix-vector operations are heavily involved. For vector-vector operations, it is not suitable for MPI parallelism as the communication takes longer than the calculation itself [3]. However, OpenMP can take over them as there is no such communication. Matrix-vector multiplication is parallelised with MPI. In addition, substantial code optimisation is carried out to improve the performance further. For some frequently used code segments, memory and file input/output operations are carefully examined.

4. Application simulation and initial performance

The development of the parallel code is demonstrated by simulating the injection of carbon dioxide in a coal seam associated carbon capture and sequestration (CCS) technologies. The simulation considers carbon dioxide injection into a 500 m radius, two-dimensional, axisymmetric, hypothetically isolated coal bed. This example has a very large domain, which is discretised by about 20,000 triangle elements. Figure 1 shows the total wall time for system matrix build and solver. The

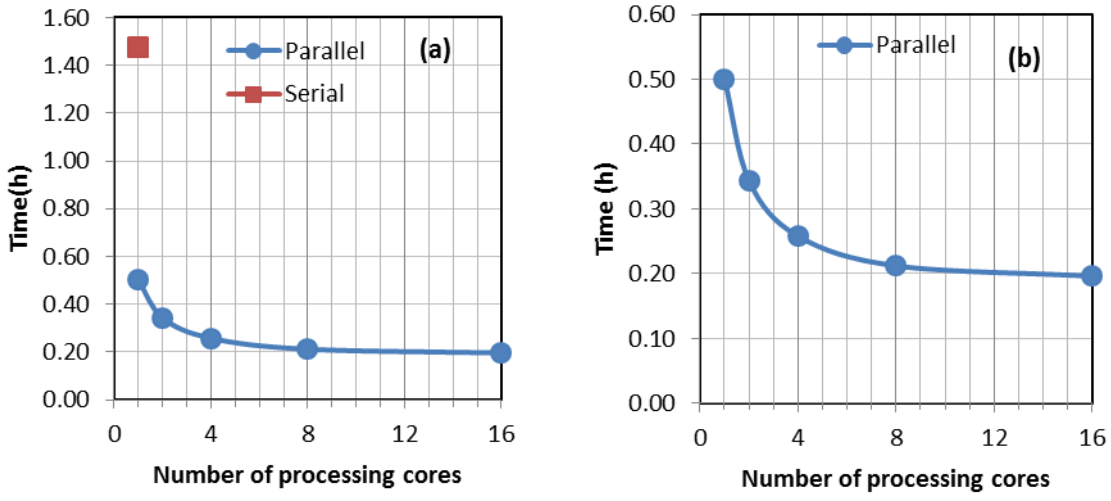

Figure 1: Total wall time of system matrix build and solver

wall time for parallel and serial modes is displayed in Figure 1(a), while the parallel results are displayed in a magnified mode in Figure 1(b). The serial code, which is not optimised, is three times slower compared with the parallel program with 1 core. Together with code optimisation, the parallel code has achieved a maximum 7.5 times speed-up compared to the non-optimised serial one.

5. Conclusions

This work here has reviewed and discussed the popular HPC hardware and parallel programming tools. Then a hybrid MPI/OpenMP model is proposed to parallelise the numerical code COMPASS. Together with code optimisation, a significant performance improvement has been seen for an example application in carbon dioxide sequestration. The developments presented in this paper will facilitate more efficient computational research in the study of carbon sequestration and other large scale geoenvironmental applications.

Acknowledgements

The work described here has been carried out as part of GRC's Seren project, which is funded by the Welsh European Funding Office (WEFO). The financial support is gratefully acknowledged. This work has been performed using the computational facilities of the Advanced Research Computing@Cardiff (ARCCA) Division, Cardiff University.

References

[1] H. R. Thomas and Y. He, A coupled heat-moisture transfer theory for deformable unsaturated soil and its algorithmic implementation, *International Journal for Numerical Methods in Engineering*, 40(18), 3421–3441, 1997.

[2] S. C. Seetharam, *An investigation of the thermro/hydro/chemical/mechanical behaviour of unsaturated soils*, Ph.D. Thesis, Cardiff University, Wales, UK, 2003.

[3] P. J. Vardon, *A three-dimensional numerical investigation of the thermo-hydro-mechanical behaviour of a large-scale prototype repository*, Ph.D. Thesis, Cardiff University, Wales, UK, 2009.

[4] S. A. Masum, *Modelling of reactive gas transport in unsaturated soil. A coupled thermo-hydro-chemical-mechanical approach*, PhD thesis, Cardiff University, Wales, UK, 2012.

[5] Top 500 supercomputer site, http://www.top500.org/.

[6] R. J. Harrison, Portable tools and applications for parallel computers, *International Journal of Quantum Chemistry*, 40, 847-863, 1991.

[7] PVM website, http://www.csm.ornl.gov/pvm/.

[8] Message Passing Interface (MPI), http://www.mpi-forum.org/.

[9] OpenMP Specifications, http://openmp.org/wp/openmp-specifications/.

[10] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, E. Aprà, Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit, *International Journal of High Performance Computing Applications*, 20, 203-231, 2006.

[11] G. D. Fletcher, M. W. Schmidt, B. M. Bode, M. S. Gordon, The Distributed Data Interface in GAMESS, *Computer Physics Communications*, 128, 190-200, 2000.

[12] M. Wang, A. J. May and P. J. Knowles, Parallel programming interface for distributed data, *Computer Physics Communications*, 180(12), 2673-2679, 2009.

[13] S. Mittal and J. Vetter, A Survey of CPU-GPU Heterogeneous Computing Techniques, *ACM Computing Surveys*, 47(4), Article 69, 2015.

[14] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, L. Dongraa, V. Eijkhout, R. Pozo, C. Romine, H. Van derVorst, *Templates For the Solution of Linear Systems: Building Blocks For Iterative Methods*, John Wiley Press, New York, 1995.