Computers & Geosciences 36 (2010) 1451–1460

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Application of a hybrid MPI/OpenMP approach for parallel groundwater model calibration using multi-core computers

Guoping Tang [a,*], Eduardo F. D'Azevedo [b], Fan Zhang [c], Jack C. Parker [d], David B. Watson [a], Philip M. Jardine [e]

[a] Environmental Sciences Division, Oak Ridge National Laboratory, P.O. Box 2008, MS-6038, Oak Ridge, TN 37831-6038, USA
[b] Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, MS-6367, Oak Ridge, TN 37831-6367, USA
[c] Institute of Tibetan Plateau Research, Chinese Academy of Sciences, P.O. Box 2871, Beijing 100085, China
[d] Department of Civil and Environmental Engineering, University of Tennessee, Knoxville, TN 37996, USA
[e] Department of Biosystems Engineering and Soil Science, University of Tennessee, Knoxville, TN 37996, USA

ARTICLE INFO

Article history:
Received 1 October 2009
Received in revised form
5 February 2010
Accepted 10 April 2010

Keywords:
Reactive transport
Coupled flow and transport
Levenberg–Marquardt algorithm
Profile
Cache

ABSTRACT

Calibration of groundwater models involves hundreds to thousands of forward solutions, each of which may solve many transient coupled nonlinear partial differential equations, resulting in a computationally intensive problem. We describe a hybrid MPI/OpenMP approach to exploit two levels of parallelisms in software and hardware to reduce calibration time on multi-core computers. HydroGeoChem 5.0 (HGC5) is parallelized using OpenMP for direct solutions for a reactive transport model application, and a field-scale coupled flow and transport model application. In the reactive transport model, a single parallelizable loop is identified to account for over 97% of the total computational time using GPROF. Addition of a few lines of OpenMP compiler directives to the loop yields a speedup of about 10 on a 16-core compute node. For the field-scale model, parallelizable loops in 14 of 174 HGC5 subroutines that require 99% of the execution time are identified. As these loops are parallelized incrementally, the scalability is found to be limited by a loop where Cray PAT detects over 90% cache missing rates. With this loop rewritten, similar speedup as the first application is achieved. The OpenMP-parallelized code can be run efficiently on multiple workstations in a network or multiple compute nodes on a cluster as slaves using parallel PEST to speedup model calibration. To run calibration on clusters as a single task, the Levenberg–Marquardt algorithm is added to HGC5 with the Jacobian calculation and lambda search parallelized using MPI. With this hybrid approach, 100–200 compute cores are used to reduce the calibration time from weeks to a few hours for these two applications. This approach is applicable to most of the existing groundwater model codes for many applications.

Published by Elsevier Ltd.

1. Introduction

Fate and transport of contaminants in the subsurface are controlled by coupled hydrological, geochemical and biological processes. HydroGeoChem 5.0 (HGC5), a three-dimensional model for fluid flow, thermal and solute transport, and biogeochemical kinetic/equilibrium reactions in saturated/unsaturated porous media (Yeh et al., 2004, 2010), is widely used to investigate contaminant migration at Department of Energy Oak Ridge Integrated Field-scale Subsurface Research Challenge (ORIFRC) site (http://www.esd.ornl.gov/orifrc/) in Tennessee (Zhang et al., 2008a) and other sites (Scheibe et al., 2009; Mayes et al., 2009). Simulations with HGC5, as well as other groundwater model codes, are often computationally intensive, particularly when multiple processes are coupled with many biogeochemical reactions. The computational time increases substantially when temporal and spatial discretization need to be refined to ensure convergence and accuracy for long-term simulations over large spatial domains.

The time increases further for inverse solutions to identify and quantify multiple processes (Gwo and Yeh, 1997; Zhang et al., 2008a). The Levenberg–Marquardt (LM) algorithm, which is widely used for model calibration, involves an iterative solution based on a gradient search procedure. Each iteration requires $2N+1$ forward model runs to compute the Jacobian matrix, when it is approximated by a central difference or $N+1$ if a forward difference is used, where N is the number of calibrated parameters (Doherty, 2004). Depending on N, initial parameter values, and nonlinearity, dozens of iterations may be required to achieve convergence, and multiple calibration runs may be

* Corresponding author. Tel.: 1 865 574 7314; fax: 1 865 576 8646.
E-mail address: tangg@ornl.gov (G. Tang).

0098-3004/$ - see front matter Published by Elsevier Ltd.
doi:10.1016/j.cageo.2010.04.013

needed to avoid local minima in the objective function. Further consideration of hybrid local/global (Sayeed and Mahinthakumar, 2005), global optimization (Vrugt et al., 2008), uncertainty analysis and multi-model comparison (Tang et al., 2009) can involve thousands of forward runs, making parallel computing necessary.

While High Performance FORTRAN was used for 3DMURF and 3DMURT (D'Azevedo and Gwo, 1997), PFEM (West and Toran, 1994) and PHGC3D (Gwo and Yeh, 1997), Open specifications for Multi-Processing (OpenMP) and Message-Passing Interface (MPI) have since emerged as the standard parallelization paradigms. OpenMP is easy to program, and facilitate increment parallelization. It was used to parallelize HBGC123D (Gwo et al., 2001), RT3D (McLaughlin, 2008), and PCG solver in MODFLOW (Dong and Li, 2009). Reasonable OpenMP performance for many single-core processors (i.e., cores) on shared-memory parallel computers (SMP), for example, 32 in Gwo et al. (2001), 64 in Hoeflinger et al. (2001) and Chapman et al. (2008), and 144 in Brown and Sharapov (2007), are reported in the literature. While the effort for each parallelization increment may not decrease, the speedup return diminishes as more increments are parallelized, reaching a "point of diminishing returns" (Hoeflinger et al., 2001). OpenMP was originally designed for expensive SMPs, and the scalability is limited by memory bandwidth.

MPI is widely used to parallelize existing software (TOUGH, Zhang et al., 2008b) and to develop new software, e.g., PARFLOW (Ashby et al., 1994) and PFLOTRAN (Mills et al., 2007), to run on massively parallel computers with hundreds of cores for TOUGH and thousands of cores for PFLOTRAN. Due to excellent scalability potential, distributed computers and MPI have been replacing SMPs and OpenMP for large-scale applications. However, MPI requires users to rewrite a serial code into a domain decomposed program (Zhang et al., 2008b).

Since the release of the first dual-core processor by IBM in 2001, Sun in 2004, and AMD in 2005 (Terboven et al., 2008), more and more (2–8) cores are built into a single processor with the development of multi-core technology. A workstation or a compute node on a cluster may have 2–4 multi-core processors with OpenMP support. OpenMP can be used to parallelize HGC5, as well as other codes, to utilize 8–16 cores on a compute node. Since multiple slaves can run on multiple workstations in a network, or multiple compute nodes on a cluster, parallel PEST (a parallel version of the widely used parameter estimation code, Doherty, 2004) can be used to utilize multiple compute nodes/workstations for model calibration. However, the number of slaves that can run simultaneously on a cluster, which is often shared by many users, is limited by its schedule policy. Embedding MPI-parallelized LM in HGC5 will enable model calibration to be run on clusters as a single job, reduce startup overhead for forward runs (Sayeed and Mahinthakumar, 2005), and eliminate the need for shared storage and associated I/O for slaves in parallel PEST. This hybrid MPI/OpenMP approach exploits two levels of parallelisms in software (coarse-grained parallelism in Jacobian calculation and lambda search in LM and fine-grained parallelism in HGC5) and in hardware (compute nodes with multiple cores). As a result, multiple compute nodes can be used efficiently for model calibration, while the ease of use of OpenMP is exploited. While hybrid MPI/OpenMP was used to exploit the coarse- and fine-grained parallelisms in a transport code (e.g., Mahinthakumar and Saied, 2005), as well as in multilevel parallelization of an inverse code (Sayeed and Mahinthakumar, 2005), we use pure OpenMP for the forward solution for ease of use and MPI-parallelized LM to extend the scalability to multiple compute nodes.

The objective of this work is to parallelize HGC5 using OpenMP and add MPI-parallelized LM to speedup groundwater calibration on clusters. We choose two ongoing simulations at ORIFRC for discussion. The first is a reactive transport simulation with >100 species and $\sim$100 reactions. The second involves simulation of field-scale coupled flow and nitrate transport. As OpenMP was used to parallelize groundwater model codes (Gwo et al., 2001), its application was relatively limited due to scalability limitation and limited access to SMPs. With the development of multi-core technology in the past several years, the interest in OpenMP has increased significantly (McLaughlin, 2008; Dong and Li, 2009). There is now great need to understand how OpenMP can be easily applied to speedup groundwater model computing efficiently on multi-core computers. We will address this need by describing how performance tools are used to facilitate the identification of the most time-consuming parallelizable loops and performance bottlenecks to minimize parallelization effort, while maximizing speedup. While previous efforts attempted to parallelize their codes for general applications (Gwo et al., 2001), we will focus on our two specific applications to illustrate that different applications can have very different parallelisms, therefore, involves different parallelization strategies and efforts, and to show the strength and limitation of OpenMP and the hybrid approach for different groundwater model applications. We hope that our work provides useful information for groundwater modelers to use multi-core computers to improve the efficiency of groundwater modeling effort.

2. Method

2.1. Parallelization using OpenMP

HGC5 solves head-based Richards equation for flow, the convection dispersion equation (CDE) for transport, heat equation for thermal transport, and mixed kinetic/equilibrium geochemical reaction equations. The Galerkin finite element and finite difference are used for spatial and temporal discretization. The nonlinear algebraic/ordinary differential equations for reactions are linearized by Newton–Raphson method, and solved by Gauss elimination with full pivoting. The flow, transport, heat transfer and reaction equations can be coupled (Yeh et al., 2004, 2010).

According to the Amdahl's law (Chapman et al., 2008), the possible speedup is approximately

$$S=\frac{1}{f/n+1-f}, \tag{1}$$

where f is the fraction of the time spent in the parallelized portion of the code when it is run in serial, and n is the number of cores. If one subroutine takes 95% of the time, parallelization of this subroutine may yield a speedup of about 9 on a 16-core compute node. As more cores are used, the speedup approaches an asymptotic value of $1/(1-f)$, e.g., 20 in the foregoing case. Because of overhead involved in starting up OpenMP library and threads, and waiting for synchronization among different cores (load imbalance), memory coherence maintenance and memory bandwidth limitation, the actual speedup is less than what is predicted by Eq. (1).

While HGC5 consists of a main routine, 174 subroutines and 2 functions (Yeh et al., 2004), only some subroutines, for example, finite element assembly, solver, particle tracking, and reaction equations (Gwo et al., 2001), material properties updating (Zhang et al., 2008b), consume most of the computational time, depending on the application. To maximize speedup and minimize effort, we profile (Chapman et al., 2008) the code for the specific application to identify the subroutines (loops) that take most of the time. Optimizing these loops may result in speedup even for a run in serial. Adding OpenMP compile directives to parallelize these loops can yield speedup for run on parallel computers. This is mostly