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 ~ 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 performed on the outer loop because inner loops are automatically run in parallel if the outer loop is in parallel.

The time-stepping loops and iteration loops are not parallelizable because the next time step and iteration are dependent on the current time step and iteration. Many other time-consuming loops, such as loops over nodes (e.g., for reactive speciation calculations and linear matrix solver), loops over elements (e.g., finite element assembly, property updating), loops over species for reactive transport, and loops over boundary sides, are parallelizable. Some loops require little or no effort to be separated into independent parallel tasks ("embarrassingly parallelizable," Chapman et al., 2008), for example, reaction calculations for each node at each iteration step (Gwo et al., 2001; McLaughlin, 2008). Other loops may need more effort. For example, for the loop over each element to assemble the global matrix and vector, updating entries corresponding to shared nodes by neighboring elements by multiple cores simultaneously may lead to unexpected results ("race condition," Chapman et al., 2008). For these loops, we use a color scheme (Gwo et al., 2001; Tracy, 2004; Mahinthakumar and Saied, 2005) to separate elements, nodes, and boundary sides into a number of groups, so that they do not share nodes with each other in a group and can be executed in multiple cores simultaneously. In more challenging cases, efforts may be involved to detect and remove hidden data dependency to ensure correctness of the numerical results (Chapman et al., 2008).

The ease to use OpenMP comes with a risk of performance. When the performance is poor, we consider changes in memory access and loop optimization to improve the serial processing, and synchronization reduction, parallel region maximization, schedule adjusting, etc., to improve parallel processing (Chapman et al., 2008). We also consider rewriting time-intensive sections of the code, and using efficient algorithms (e.g., solvers).

In summary, the parallelization procedure for one increment is:

- **profile** the code to find the most time-consuming parallelizable outer loop;
- **parallelize** the loop by adding OpenMP compiler directives and check the correctness by comparing results with the code without this parallelization;
- **analyze** the parallel performance and **tune** the code if not satisfied.

This procedure is repeated incrementally until the performance is satisfied.

2.2. Parallelization of Levenburg–Marquardt algorithm using MPI

It is widely recognized that most of the time in an LM is spent on the forward runs for the Jacobian calculation and lambda search. LM is mostly parallelized using a master/slave model (Doherty, 2004; Cao et al., 2009), in which the master controls the execution and the slaves and the master run the forward predictions. We follow Cao et al. (2009) with minor modifications (Fig. 1). For N adjustable parameters, we use $N+1$ compute nodes if a forward difference is used to approximate Jacobian. If a central difference is used, we appropriate $2N+1$ compute nodes. All compute nodes are used in the lambda search processes. Note that this approach can be easily extended to global or hybrid local/global optimization schemes or Monte Carlo analysis algorithms.

2.3. Computers and tools

Simulations are performed on Jaguar XT5 and Lens at National Center for Computational Sciences (NCCS, www.nccs.gov/computing-resources), ORNL Institutional Cluster (OIC, http://oic.ornl.gov), and a 4-core workstation. The publicly available utility codes GFORTRAN, GDB, and GPROF (Fenlason and Stallman, 2000) are used for compiling, debugging, and profiling the FORTRAN code. We also insert code to clock loops for performance analysis and tuning. While the performance for the forward runs is similar among these computers, the results from Lens are presented, because it has the maximum number of cores (16). The inverse

```
Read data for HGC and LM (master only, MO);
MPI_BCast data for HGC and LM (master and slaves, MS);
--optimization loop-----------------
  Perturb parameters ($b$) for Jacobian calculation (MO);
  MPI_Scatter b (MS);
  Run subroutine HGC5 to obtain predictions (MS);
  Calculate residuals ($r$) and objective function ($\varphi$) (MS);
  Gather predictions(MS).
  Calculate Jacobian J, $J^tQJ$ and $J^tQr$ (MO).
  If $|J^tQr|$ < tol1, done ($Q$ is observation weight)(MO).
----lambda search loop------------
    Create lambda ($\lambda$) values (MO);
    MPI_BCast  $J^tQJ$ and $J^tQr$ (MS);
    MPI_Scatter $\lambda$ (MS);;
    Solve $(J^tQJ+\lambda I)\delta b = J^tQr$ (MS);
    MPI_Reduce maximum $|\delta b|$(MS);
    If $|\delta b| < tol2|b|$, done (MO);
    Adjust $b$ in bounds (MS);
    Run HGC5 subroutine, calculate $r$ and $\varphi$ (MS);
    MPI_Gather $\varphi$ (MS);
    Find minimum $\varphi$ (MO);
    If $\varphi$ decreases, repeat $\lambda$ search loop until MAXLBD (MO)
----lambda search loop------------
  If $\varphi$ decreases, update $\lambda$, $b$&$\varphi$, repeat optimization loop
until MAXOPT (MO);
--optimization loop-----------------
Postprocess (MO)
```

Fig. 1. MPI parallelization of Levenberg–Marquardt algorithm.

simulations are run on Jaguar XT5, due to availability of the needed cores (100–200). Because the memory-use pattern is critical for OpenMP code performance, we use Cray PAT (NCCS website) to check memory usage (e.g., cache missing rate, Chapman et al., 2008) and load balance on Jaguar XT5. The operating system is Redhat Linux.

3. Applications

We select two problems from ongoing multi-scale multi-process research at ORIFRC. In the first application, HGC5 is used to simulate transport of uranium species in a column packed with sediment involving base addition to immobilize uranium. The purpose is to quantify the chemical reactions that control uranium precipitation and sorption in the subsurface as pH is raised. In the second application, HGC5 is used to simulate the evolution of a nitrate plume over a fifty-year period. The objective for this work is to speedup the simulations by parallelizing HGC5 and LM.

3.1. Uranium transport in column with pH manipulation

The column has a diameter of 51 mm and length of 380 mm. The simulation consists of 94 equilibrium reactions: 72 aqueous complexation reactions, 22 soil pH buffering, ion and cation exchange reactions, and 5 kinetic precipitation reactions involving 141 species (Zhang et al., 2008a). A uniform grid of 47.5 mm (8 hexahedral elements, 32 nodes) is used for spatial discretization, and a uniform time step of 0.4 h is used for the 33 day simulation duration. The model is calibrated against concentration measurements over time for 12 species to estimate 12 reaction parameters related to the kinetic precipitation reactions and equilibrium soil buffering reactions. A typical forward run takes 20–30 min on a workstation, and the parameter estimation takes several days.

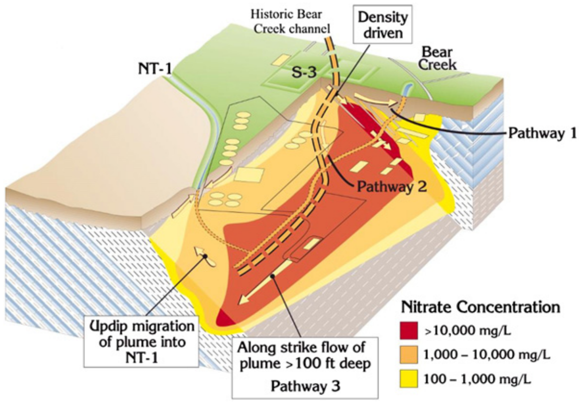

Fig. 2. Conceptual model for nitrate transport under S-3 ponds (SAIC, 1997).

3.2. Field-scale groundwater flow and nitrate transport problem

The ORIFRC site is located in Bear Creek Valley on the DOE Oak Ridge Reservation in Tennessee. Groundwater beneath the site is contaminated with nitrate, radionuclides, and other metals resulting from the operation of the S-3 Waste Disposal Ponds for over thirty years since 1951. The contaminants migrate along the strike to the west, along the dip into the subsurface, and across the bedding to Bear Creek (Fig. 2). A groundwater model is constructed to investigate the transport of nitrate in the Nolichucky shale formation using HGC5 (Fig. 3).

The grid consists of 10,323 triangular prism elements and 6320 nodes. In addition to saprolite at the surface and bedrock at depth, a transition zone, a trench, a gravel fill and a deeper high permeability zone are included in a spatial domain that is about 275 m deep, 1400 m long and 500 m wide. For the flow simulation, constant head boundary conditions are estimated through interpolation from observed water levels and a regional groundwater model for the streams and the lateral perimeter of the domain (JE, 1996). Recharge is applied at the ground surface. The S-3 ponds were capped in 1985 when a no flow condition is applied. For the transport simulation, a zero concentration gradient condition is assumed on the model perimeter and at streams. No flux is assumed at the bottom, and a prescribed mass flux is prescribed at the S-3 ponds area during the period of operation from 1951 to 1983.

The initial condition is assumed to be steady state with rainfall recharge for flow and no contaminants for transport. The flow and transport equations are coupled to account for density-driven flow due to high nitrate concentrations. A full tensor is used for the hydraulic conductivity to account for anisotropy in the dipping geological formations. Two exponential depth reduction factors are assumed to approximate decreases in rock porosity and hydraulic conductivity with depth. A first order decay coefficient is used to approximate natural attenuation rate of nitrate. Monitoring data for water levels (151) and nitrate

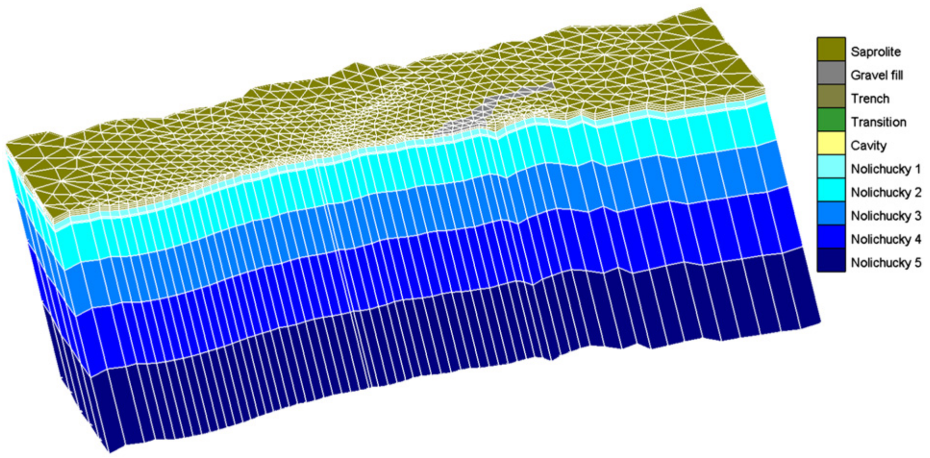

Fig. 3. Groundwater model to simulate nitrate migration under S-3 ponds.

concentrations (1114) from 195 monitoring wells are used for calibration. A forward run takes several hours to a day depending on the time step, convergence tolerance, and the parameter values. With 11 calibration parameters, a calibration run takes over a week or longer using PEST on a workstation depending on time stepping and convergence tolerance.

4. Results and discussion

4.1. Uranium transport in column with base addition

Profiling with GPROF yields the time spent in each subroutine, excluding time spent in "children" (subroutines called by the subroutine, Fig. 4) in the flat profile (Fenlason and Stallman, 2000). Over 80% of the time is spent in DGELG, which is called by KINEQL to solve the linearized reaction equations, and more than 93% is spent in the first four subroutines. All of them are directly or indirectly called by BIOGEOCHEM (KEMOD in Yeh et al., 2004) for the reaction calculations. Other subroutines with notable time are Q468D and Q468 for numerical integration in flux calculation (MVELT) and finite element assembly (ASEMBL).

We trace from the time-intensive subroutines back to their parents, using the call graph (Fenlason and Stallman, 2000) output by GPROF (Fig. 5). BIOGEOCHEM is called by OCSPIT in a reaction loop over each node; OCSPIT is called by CHEMIT in an iterative transport loop over reactions (OCSPIT) and transport (ASEMBL and TRANSP); CHEMIT is called in a time loop in HGC50; and HGC50 is called by the main program. While the last two loops are not parallelizable, the reaction calculation in each node is independent of other nodes in the reaction loop, therefore, is "embarrassingly parallelizable" (Gwo et al., 2001; McLaughlin, 2008). We identify this loop as the first parallelization increment. According to Eq. (1), parallelization of this loop may result in speedup of 11 for 16 cores, since it takes 97% of the time.

Alternatively, we can identify this increment by examining the call graph (Fig. 5) from top to bottom. In the main program, the I/O and data preparation takes little time, while over 99% of the time is spent in HGC50 and its children. CHEMIT, MVELT, and their children take 98.3%+1.4% of the total time, and they are called in a time loop that is not parallelizable. CHEMIT solves CDE using ASEMBL and TRANSP and reaction equations using OCSPIT iteratively. OCSPIT consumes 97% of the total time, and the loop over each node calling BIOGEOCHEM is parallelizable. Similarly,

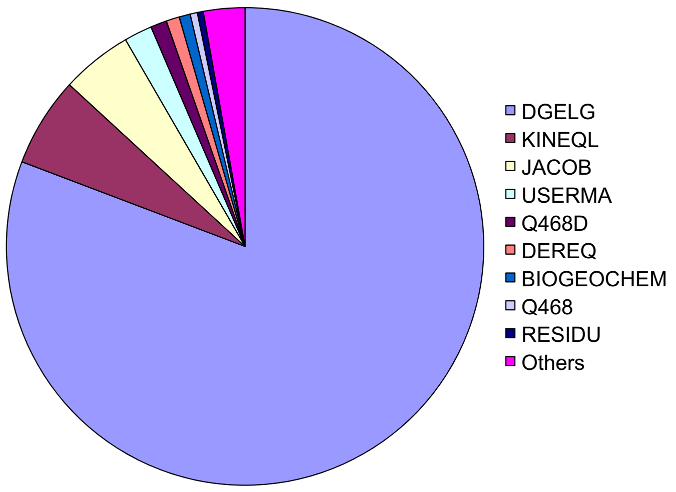

Fig. 4. Computation time distribution among subroutines (based on flat profile produced by GPROF Fenlason and Stallman, 2000), for reactive transport problem.

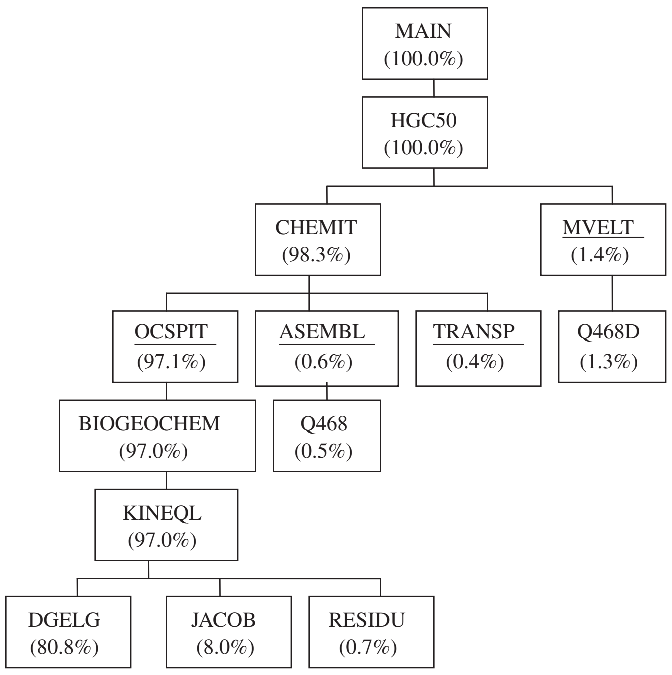

Fig. 5. Call graph describing calling relationships among subroutines and computing times (percentage in brackets) in each subroutine and its children for reactive transport problem (based on call graph output by GPROF). Subroutines selected for parallelization are underlined. Subroutines that spend an insignificant amount of time are not included. Among subroutines, CHEMIT solves transient reactive transport equations, OCSPIT calls BIOGEOCHEM to solve ordinary differential and nonlinear algebraic reaction equations, ASEMBL assembles matrix equation for transport, TRANSP solves transport equation considering reactions, and MVELT computes solute flux.

```
!$OMP PARALLEL DO
!$OMP&PRIVATE(private variable list)
      DO NP=1,NNP
      ......
      CALL BIOGEOCHEM(argument list…)
      ......
      END DO
!$OMP END PARALLEL DO
```

Fig. 6. Adding OpenMP compiler directives to parallelize loop in an OCSPIT.

we identify the loops in MVELT, TRANSP, and ASEMBL as other increments.

In the first increment, the "parallel do" compiler directive is added to instruct the compiler to create a number of to use multiple cores to execute the loop in parallel (Fig. 6). In order for the parallelization to work correctly, we have to set the correct scope for each of the variables in the loop. For each of the 111 arguments in BIOGEOCHEM call statement, we identify whether it is shared among different loops (shared variable) or not (private variable), how it is initialized at the beginning of the loop and returned at the end of the loop, and whether it is revised in the loop, and explicitly list the private variables in the compile directive. With this increment, the speedup is about 10 for 16 cores (Fig. 7), which is not as high as predicted by Eq. (1), because there are only 36 nodes. While the first 32 loops are distributed evenly to the 16 cores, eight cores will be idle while the other four cores work on the last four loops (load imbalance).

In the second increment, we parallelize the loop in MVELT, which calculates material fluxes due to dispersion and advection for each component at each node (Yeh et al., 2004). It contains an outer loop over each component species (42), a loop over each element (8), and several inner loops over each node. Because the flux calculations for each component are independent from others, we add compiler directives to the outer loop, and explicitly list private variables. For better use of memory, and loop fusion for better performance, we move the initialization statements into the outer loop. This increment results in a slight improvement in speedup (Fig. 7), because MVELT consumes only 1% of the total time.

In the third increment, we parallelize TRANSP, which solves CDE for each primary dependent variable (Yeh et al., 2004). The outer loop operates over 42 components and 5 kinetic variables and is "embarrassingly parallelizable". CDE is really solved for only 18 mobile components and 1 mobile kinetic variable. We form a list of these variables, and modify the loop to improve load balance. With this increment, the speedup is further improved (Fig. 7).

Finally, we parallelize ASEMBL for finite element assembly for CDE. The entries in a conductivity or mass matrix corresponding to the shared nodes among neighboring elements may be updated in multiple loops/cores simultaneously, so the results can be incorrect (race condition, Chapman et al., 2008). We simply partition the elements into two groups: one for odd number elements and one for even number elements, each with four elements. We replace the loop with an outer loop over each group and an inner loop over each element (Tracy, 2004). Because the elements in each group do not share any nodes with other elements, the inner loop is parallelizable. We add an OpenMP compiler directive to the inner loop, and the speedup is a little bit increased (Fig. 7).

In summary, using GPROF, parallelizable time-consuming subroutines/loops are identified without delving into the details of the HGC5 code. With a few lines of OpenMP compiler directives added to the loop in OCSPIT, the speedup increases to over 10 for 16 cores in this application. Even though setting correct scopes for the variables in this loop to avoid race condition can involve significant effort, in general we can focus on this loop and ignore the rest of the code. Significant speedup can be achieved without a thorough understanding of a complex code. This is a great advantage of using OpenMP to incrementally parallelize a sophisticated code for an application when only a few parallelizable loops are highly time intensive. Speedup increases as MVELT, TRANSP, and ASEMBL are parallelized, but the gain decreases with increasing increments (diminishing return, Hoeflinger et al., 2001). The performance is not as good as expected by Eq. (1) when 8 or 16 cores are used due to load imbalance: there are only four elements in each group, so that the time taken by ASEMBL does not decrease when more than four cores are used (Fig. 8); we have 36 nodes in OCSPIT, 42 mobile species in MVELT, and 19 mobile variables in TRANSP, load imbalance increases when more cores are used.

To further examine the load imbalance, and to check if the spatial grid is fine enough, we refine the model to use 19 elements (80 nodes). The parallel performance is much better than the 8-element case (Fig. 7). It is excellent for OCSPIT because the computation for 80 nodes is evenly distributed to 8 or 16 cores (Fig. 8). The speedup for ASEMBL improves a little, since there are 10 and 9 elements in the two groups. The speedup for MVELT and TRANSP remains the same because the load imbalance does not improve. Performance analysis with Cray PAT on Jaguar confirms the load imbalance, and shows that the cache missing rate (Chapman et al., 2008) is less than a few percent. The reason is that most of the time is spent in BIOGEOCHEM, and only local small arrays are used in the computation. Clearly, the parallel performance for this application is limited by the simulation itself (small scale), and OpenMP works very well for this application.

To check temporal discretization error, the time step is reduced to 0.1 h. While there is little difference in the predictions for pH, Al, Si, and NO_3, differences for other species are evident and increase with time as more base is added (Fig. 9). With reduced computational time, we can increase resolution to improve accuracy of the numerical solutions to increase our confidence.

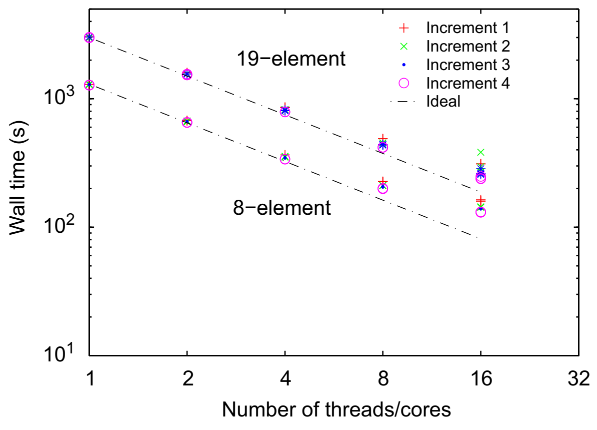

Fig. 7. Speedup for reactive transport application with increasing parallelization increments (subroutines OCSPIT, MVELT, TRANSP, and ASEMBL) for two grid resolutions (8 elements versus 19 elements). Wall (clock) time is real time elapses from start to end of program run, i.e., an actual time taken by a computer to complete a task. Each run is repeated three times to clock wall time.

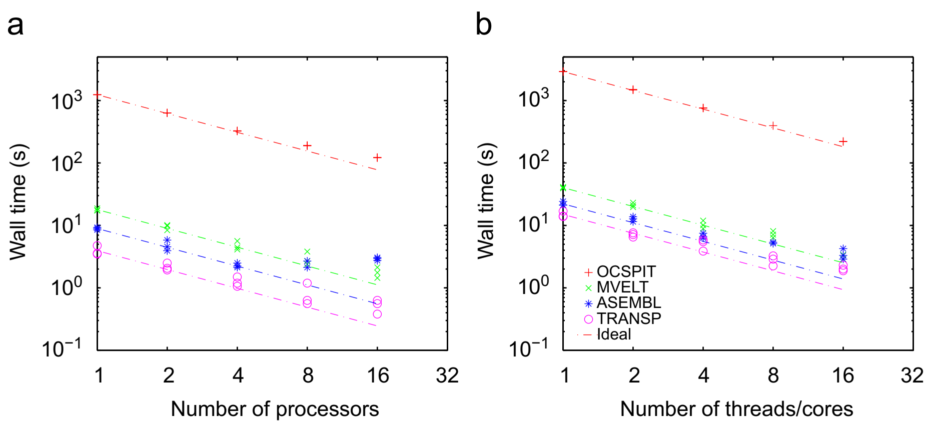

Fig. 8. Speedup for subroutines parallelized in reactive transport problem. OCPIT, MVELT, ASEMBL, and TRANSP are subroutines that are parallelized. Timing measurements are repeated three times to show variations that are apparent for TRANSP, because number is small.

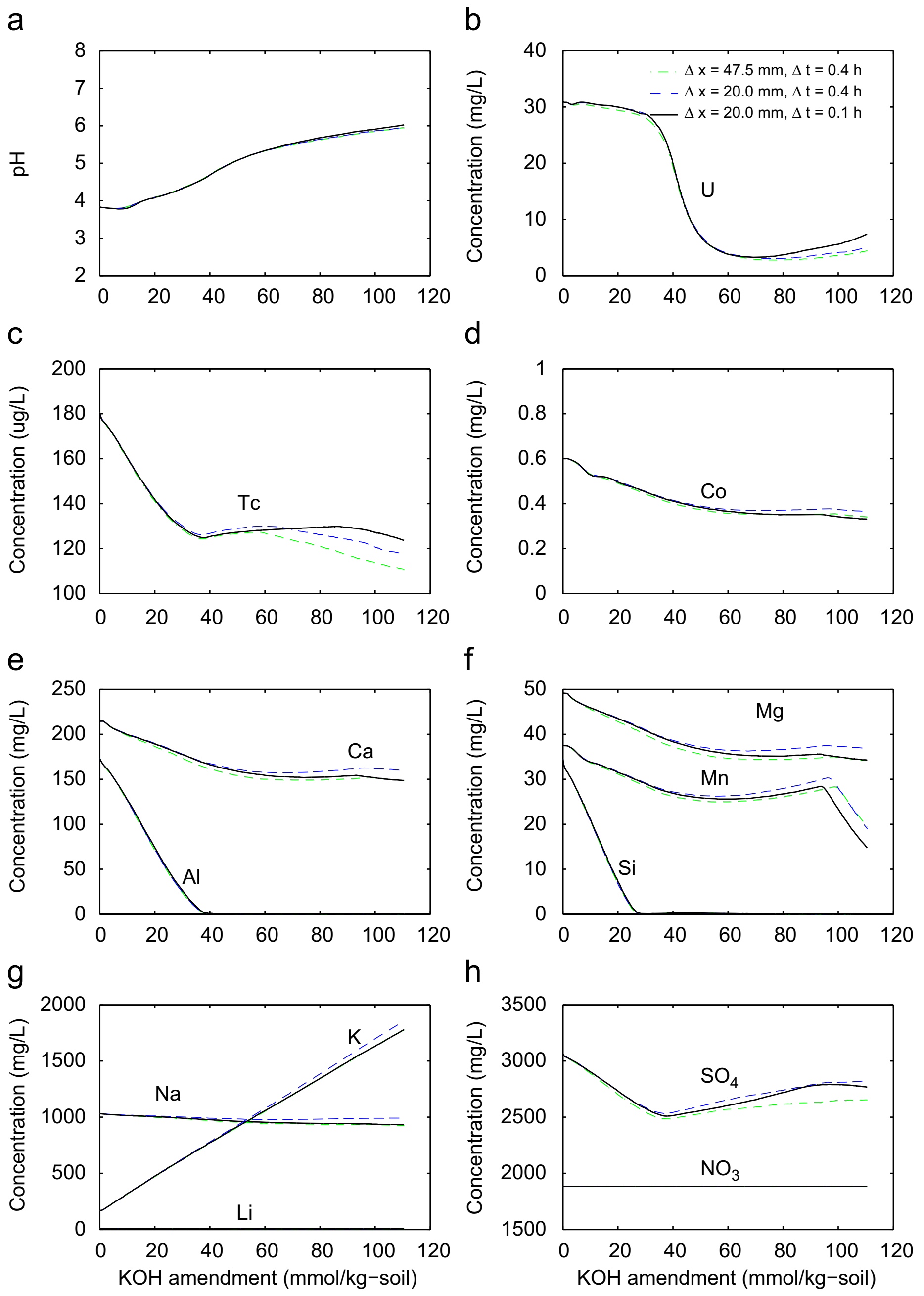

Fig. 9. Concentration prediction comparison for major species between coarse and fine spatial and temporal discretization.

For the calibration, we estimate 12 parameters using 392 observations. Because each iteration involves 13 (forward difference) or 25 (center difference) forward runs for the Jacobian calculations plus a number of runs for lambda searches and each parameter estimation run involves multiple iterations, parameter estimation using PEST and HGC5 on a workstation takes several days. Using parallel PEST, we run a master and 12 slaves on 13 Original OIC nodes, the computational time is reduced to several hours. By running the OpenMP-parallelized HGC5 code on each node with two threads for the two cores, the time is further cut by half.

While each compute node on Phase-2 OIC, Jaguar XT5, and Lens has 8, 8, and 16 cores, respectively, the schedule policy makes it impractical to run more than a few slaves with parallel PEST. We ran the hybrid MPI/OpenMP code with 13 (forward difference) and 25 (center difference) compute nodes (104 and 200 cores) on Jaguar XT5. With each forward run requiring less than 5 min now, over 24 optimization or lambda search iterations can be finished within a 2 h limit. Most of the computing time is spent on the forward solution. The variation of compute time for different compute nodes is small, resulting in good load balance. The hybrid MPI/OpenMP approach greatly reduces the computing time for this application from days to a few hours.

4.2. Field-scale groundwater flow and nitrate transport problem

To save computational time and make the solution robust for calibration, we employ an automatic time-stepping based on the iterations to solve the coupled flow and transport in each time step. The time-step starts from an initial value $\Delta t_o = 1$ day and changes after each time-step to min(Δt_{max}, $\Delta t + \alpha\Delta t$), where Δt is

Table 1
Distribution of computational time among parallelizable subroutines for the field-scale flow and transport application.

Subroutine	% Time	Function
ASEMBL	25.16	Finite element assembly for transport
JPCG	23.19	Jacobi preconditioned conjugate gradient solver for flow
VELT	14.89	Velocity field calculation
FASEMB	10.72	Finite element assembly for flow
SPROP	6.52	Soil property (moisture content, conductivity, and capacity)
BICGSTAB	5.36	BiCGSTAB solver for transport
AFABTA	3.14	Upstream weighting factor
GRDROV	2.47	Gradient term of flux
OCSPIT	2.44	Reaction equation solver
RHOMU0	1.98	Ratio of fluid density and viscosity over reference value
DISPC	1.12	Hydrodynamic dispersion coefficient
EQNGEN_P	0.94	Transport matrix equation generation
NODVA2	0.59	Moisture content value from element to node
FSFLOW	0.47	Boundary water flux and domain water increase
Others	1.01	

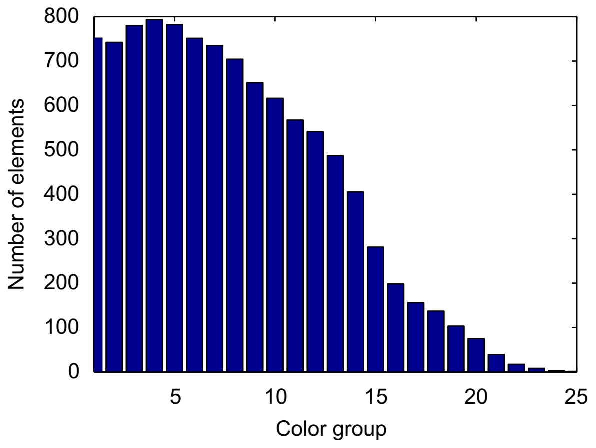

Fig. 10. Number of elements in each color group.

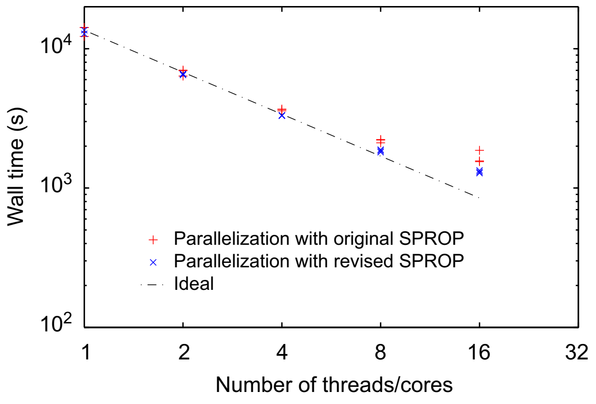

Fig. 11. Speedup for field-scale flow and transport problem. Timing measurements are repeated three times to show variability.

the current time-step size, α is a multiplier, and $\Delta t_{max}=30$ days. When the number of iterations for convergence is less than 4, the next time-step is increased by 5% ($\alpha=0.05$); when the number of iterations exceeds 10, the next time-step is reduced to 0.8 Δt ($\alpha=-0.2$). To solve the linear equations with sparse matrix of 6320×30 efficiently, the conjugate gradient solvers with Jacobi preconditioning (JPCG, BICGSTAB) are added for the ease of parallelization (Barrett et al., 1994).

We again use GPROF to profile the code to identify parallelizable time-intensive subroutines (Table 1). Comparing with the first application, computational time is spread over a greater number of subroutines. Nevertheless, over 99% of the time is spent in less than 14 subroutines. While parallelization of OCSPIT is similar, parallelizing the loop over species in TRANSP will not help in this application, because there is only one species (iteration) in the loop. For the finite element assembly, as well as VELT and NODVA2, dividing the elements into independent groups is more complicated than in the first application. We use the algorithm for a color scheme (see Tracy, 2004 for details) to partition the elements into 25 groups. The number of elements is greater than 100 in the first 19 groups, and decreases with increasing group index in the rest (Fig. 10). Load imbalance is not expected to be significant as we use 8–16 cores. The color scheme is also used to separate the boundary sides and nodes for the application of boundary conditions (BC, FBC), and boundary flux calculation (FSFLOW). The loops in the rest of these subroutines, SPROP, RHOMU0, DISPC, AFABTA, GRDROV, EQNGEN_P, and FBCMTRX, are embarrassingly parallelizable. All of these subroutines are parallelized incrementally, using OpenMP similar to the first application.

This parallelization scales well up to 8 cores, but somewhat deteriorates for 16 cores (Fig. 11). Checking the performance of the parallelized subroutines, there is little speedup for SPROP for more than 4 cores; speedup for JPCG, BICGSTAB, and OCSPIT deteriorates for more than 8 cores; while other subroutines perform reasonably well (Fig. 12). SPROP updates water content, water storage, hydraulic conductivity tensor, and effective porosity based on nodal pressure heads and coordinates, density and viscosity at each Gauss point, and soil property parameters and element-node connection information. All of these variables are stored in large arrays. For example, the array for hydraulic conductivity is 10,323 (elements) $\times$ 8 (Gauss points) by 6 (components for hydraulic tensor), which uses about 4 MB memory, while each core has 64 KB L1 and 512 KB L2 dedicated cache and each node has 2 MB L3 shared cache on Jaguar XT5. Accessing so many large arrays in a loop is not cache efficient. Cray PAT reveals over 90% L2 cache missing rate as it runs on Jaguar XT5. We rewrite this subroutine to simplify it as much as possible to reduce execution time. Even though the speedup remains poor for SPROP (Fig. 12c), the overall performance is significantly improved (Fig. 11).

The solver computational time is mostly spent on the matrix vector multiplication with the matrix stored in a compressed form (Yeh, 1985). The matrix is defined as CMATRX(6320,30), which occupies about 1.5 MB memory. It is accessed within an outer loop over rows and an inner loop over columns through an index array LRN (Yeh, 1985). So the matrix vector multiplication is not cache friendly (Chapman et al., 2008). Analysis with Cray PAT shows that the L2 cache missing rate is over 85% when running in serial. When 8 cores are used, the cache missing rate decreases to below 10%, probably because there is much more cache memory. Speedup deterioration with 16 cores is likely due to overhead by the operation system to maintain coherency of the cache on the 16 cores.

We use 12 compute nodes to calibrate 11 parameters. This reduces the calibration time from days or weeks to several hours. The calibration often does not converge within the 2-hour time limit for Jaguar XT5. Since the initial parameter input file is updated after each iteration, runs can be restarted from the intermediate parameter vector to finalize the optimization. Even though the parallel scalability for this application is somewhat limited, and the parallelization effort is greater than that for the

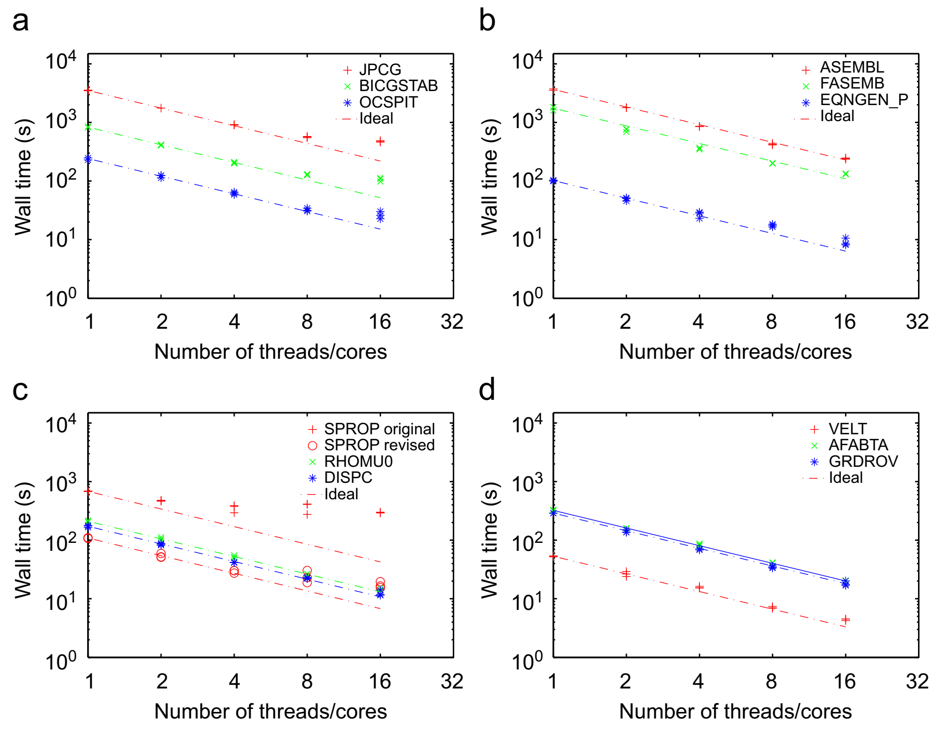

Fig. 12. Speedup for subroutines parallelized in field-scale flow and transport problem. Timing measurements are repeated three times to show variability.

first application, the hybrid approach significantly reduces calibration time with reasonable effort.

5. Summary and conclusions

HYDROGEOCHEM 5.0 is parallelized using OpenMP for a column-scale reactive transport simulation and for a field-scale coupled flow and transport simulation. In the first application, over 97% of the computational time is spent in a parallelizable loop. Adding a few lines of OpenMP compiler directives to the loop reduces computational time by a factor of 10 on a 16-core compute node. Selectively parallelizing a few more loops further improves parallel performance. For the field-scale application, most of the time is spent in loops within 14 subroutines. With addition of preconditioned conjugate gradient solvers, and use of a color scheme to partition elements, nodes, and boundary sides to avoid race conditions, loops in these subroutines are parallelized using OpenMP, resulting in a similar speedup as the first application. The performance analysis tools (GPROF, Cray PAT) are useful in the identification of time-consuming parallelizable subroutines and performance bottlenecks, so that we can maximize speedup with minimal effort by focusing on those time-consuming subroutines.

While scalability for the forward solution is limited by the number of cores on a compute node, scalability for the inverse problem is extended by employing compute nodes at the number of calibration parameters, or twice, using either parallel PEST or MPI-parallelized Levenberg–Marquardt algorithm. This hybrid approach enabled us to use hundreds of compute cores to reduce calibrating time from days or weeks to a few hours. This approach can be easily extended for global or hybrid local/global optimization, or uncertainty analyses using Monte Carlo method, where a large number of compute cores can be utilized.

The scale of the first application is small, and memory access by the time intensive reaction calculations is cache friendly. Scalability for this application is limited primarily by load imbalance. While load balance is not an issue in the field-scale application for up to 16 cores, scalability is limited by memory access. While only minor effort was needed to use OpenMP to exploit parallelism in the small scale application, the field-scale application requires significantly more effort. As the scale or the speedup requirement of the application increases, advanced domain decomposition and improved data placement are needed to maintain data locality to overcome the memory access bottleneck when use of MPI can be more advantageous.

Our applications also show that the parallelism can be very different for different applications, therefore, requires different strategies to fully exploit the diverse parallelisms. OpenMP is useful and efficient, and may serve as a stepping stone to subsequent MPI parallelization. Note that OpenMP, performance tools, hybrid MPI/OpenMP, and the methodology described are not only applicable to HGC5, but also applicable to most of the groundwater modeling codes. We hope that many groundwater modelers can benefit from this hybrid MPI/OpenMP approach, because many groundwater model applications are at scales bracketed by our two examples.

Acknowledgement

We are indebted to Dr. Jack Gwo from USNRC for his generous help. We appreciate Dr. Jeff Nichols from Environmental Sciences Division for internal review. This research was funded by the U.S. Department of Energy, Office of Science, Office of the Biological and Environmental Research, and used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy. Oak Ridge National Laboratory is managed by the UT-Battelle, LLC, under Contract DE-AC05-00OR22725 with the US DOE. Partial support was also provided by the U.S. Department of Defense Strategic Environmental Research and Development Program (SERDP) Project ER-1611.

References

Ashby, S.F., Falgout, R.D., Smith, S.G., Tompson, A.F.B., 1994. Modeling groundwater flow on MPPs. In: Skjellum, A., Resse, D. (Eds.), Proceedings of the 1993 Scalable Parallel Libraries Conference. IEEE Computer Society Press, Los Alamitos, CA, pp. 17–25.

Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhour, V., Pozo, R., Romine, C., van der Vorst, H., 1994. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.. SIAM, Philadelphia, PA 112pp.

Brown, R., Sharapov, I., 2007. High-scalability parallelization of a molecular modeling application: performance and productivity comparison between OpenMP and MPI implementations. International Journal of Parallel Programming 35, 441–458.

Cao, J., Novstrup, K.A., Goyal, A., Midkiff, S.P., Caruthers, J.M., 2009. A parallel Levenberg–Marquardt algorithm. In: Proceedings of the 23rd International Conference on Supercomputing, Yorktown Heights, NY, USA, 450–459.

Chapman, B., Jost, G., van der Pas, R., 2008. Using OpenMP: Portable Shared-Memory Parallel Programming. The MIT Press, Cambridge, MA 353pp.

D'Azevedo, E.F., Gwo, J.P., 1997. Parallelization of a multiregion flow and transport code using software emulated global shared memory and high performance FORTRAN. In: Tentner, A. (Ed.), High Performance Computing 1997: Grand Challenges in Computer Simulation, The Society for Computer Simulation, pp. 21–26.

Doherty, J., 2004. PEST: Model Independent Parameter Estimation: User Manual fifth ed. Watermark Numerical Computing, Brisbane, Australia 336pp.

Dong, Y., Li, G., 2009. A parallel PCG solver for MODFLOW. Ground Water 47 (6), 845–850.

Fenlason, J., Stallman, R., 2000. GNU GPROF—The Gnu Profiler. Free Software Foundation, Inc. 52pp. 〈www.skyfree.org/linux/references/gprof.pdf〉 (accessed 21.07.10.).

Gwo, J., D'Azevedo, E.F., Frenzel, H., Mayes, M., Yeh, G.-T., Jardine, P., Salvage, K., Hoffman, F., 2001. HBGC123D: a high-performance computer model of coupled hydrogeological and biogeochemical processes. Computers & Geosciences 27 (10), 1231–1242.

Gwo, J.P., Yeh, G.-T., 1997. Application of a parallel 3-dimensional hydrogeochemistry HPF code to a proposed waste disposal site at the Oak Ridge National Laboratory. In: Tentner, A. (Ed.), High Performance Computing 1997: Grand Challenges in Computer Simulation. The Society for Computer Simulation, Atlanta, GA, USA, pp. 128–133.

Hoeflinger, J., Alavilli, P., Jackson, T., Kuhn, B., 2001. Producing scalable performance with OpenMP: experiments with two CFD applications. Parallel Computing 27, 391–413.

JE (Jacobs Engineering), 1996. Appendix F: Regional Groundwater Flow Model Construction and Calibration. Bear Creek Valley feasibility study. Oak Ridge Y-12 Plant, Oak Ridge, Tennessee. Jacobs Engineering, Oak Ridge, TN, USA 132pp.

Mahinthakumar, G., Saied, F., 2005. A hybrid MPI-OpenMP implementation of an implicit finite-element code on parallel architectures. International Journal of High Performance Computing Applications 16 (4), 371–393.

Mayes, M.A., Tang, G., Jardine, P.M., McKay, L.D., Yin, X.L., Pace, M.N., Parker, J.C., Zhang, F., Mehlhorn, T.L., Dansby-Sparks, R., 2009. Influence of sedimentary bedding on reactive transport parameters under unsaturated conditions. Soil Science Society of America 73, 1938–1946.

McLaughlin, J.D., 2008. Parallel processing of reactive transport models using OpenMP. M.Sc. Thesis, Brigham Young University, Provo, UT, USA, 82pp.

Mills, R.T., Lu, C., Lichtner, P.C., Hammond, G.E., 2007. Simulating subsurface flow and transport on ultrascale computers using PFLOTRAN. Journal of Physics: Conference Series 78, 012051. doi:10.1088/1742-6596/78/1/012051.

SAIC (Science Applications International Corporation), 1997. Report on the Remedial Investigation of Bear Creek Valley at the Oak Ridge Y-12 plant, Oak Ridge, Tennessee, vol. 1. Department of Energy, Oak Ridge, TN, USA DOE/OR/01-1455/V1&D1, 475pp.

Sayeed, M., Mahinthakumar, G., 2005. Efficient parallel implementation of hybrid optimization approaches for solving groundwater inverse problems. Journal of Computing in Civil Engineering 19 (4), 329–340.

Scheibe, T., Mahadevan, R., Fang, Y., Garg, S., Lovley, D., 2009. Coupling a genome-scale metabolic model with a reactive transport model to describe in situ uranium bioremediation. Microbial Biotechnology 2 (2), 274–286.

Tang, G., Mayes, M.A., Parker, J.C., Yin, X.L., Watson, D.B., Jardine, P.M., 2009. Improving parameter estimation for column experiments by multi-model evaluation and comparison. Journal of Hydrology 376, 567–578. doi:10.1016/j.jhydrol.2009.07.063.

Terboven, C., an Mey, D., Sarholz, S., 2008. OpenMP on multicore architectures. In: Chapman, B., Zheng, W., Gao, G.R., Sato, M., Ayguadé, E., Wang, D. (Eds.), A Practical Programming Model for the Multi-core Era. Springer, Berlin, pp. 54–64.

Tracy, F.T., 2004. Optimizing finite element programs on the Cray X1 using coloring schemes. In: Proceedings of the Users Group Conference (DOD_UGC'04). IEEE Computer Soceity, Washington, DC, USA, pp. 313–317.

Vrugt, J.A., Stauffer, P.H., Wöhling, Th., Robinson, B.A., Vesselinov, V.V., 2008. Inverse modeling of subsurface flow and transport properties: a review with recent advances in self-adaptive global optimization, sequential data assimilation, and parallel computing. Vadose Zone Journal 7 (2), 843–864.

West, O.M., Toran, L.E., 1994. Development of a Three-Dimensional Groundwater Flow Model for Western Melton Valley: Application of P-FEM to a DOE Waste Site. ORNL/TM-12474. Oak Ridge National Laboratory, Oak Ridge, TN, USA 39pp.

Yeh, G.-T., 1985. Comparisons of successive iteration and direct methods to solve finite element equations of aquifer contaminant transport. Water Resources Research 21 (3), 272–280.

Yeh, G.-T., Fang, Y., Zhang, F., Sun, S., Li, Y., Li, M.-H., Siegel, M.D., 2010. Numerical modeling of coupled fluid flow and thermal and reactive biogeochemical transport in porous and fractured media. Computational Geosciences 14 (1), 149–170.

Yeh, G.-T., Sun, J.T., Jardine, P.M., Burgos, W.D., Fang, Y., Li, L., Siegel, M.D., 2004. HYDROGEOCHEM 5.0: A Three-Dimensional Model of Coupled Fluid Flow, Thermal Transport, and Hydrogeochemical Transport through Variably Saturated Conditions—Version 5.0. ORNL/TM-2004. Oak Ridge National Laboratory, Oak Ridge, TN, USA 255pp.

Zhang, F., Luo, W., Parker, J.C., Spalding, B.P., Brooks, S.C., Watson, D.B., Jardine, P.M., Gu, B., 2008a. Geochemical reactions affecting aqueous-solid partitioning metals during titration of uranium contaminated soil. Environmental Science and Technology 42 (21), 8007–8013.

Zhang, K., Wu, Y., Pruess, K., 2008b. User's Guide for TOUGH2-MP—A Massively Parallel Version of the TOUGH2 Code. LBNL-315E. Lawrence Berkeley National Laboratory, Berkeley, CA, USA 108pp.