

Review of the algebraic linear methods and parallel implementation in numerical simulation of groundwater flow

CHENG Tang-pei^{1,2}, LIU Xing-wei³, SHAO Jing-Li^{3*}, CUI Ya-li³

¹Institute of Applied Physics and Computational Mathematics, Beijing 100088, China.

²CAEP Software Center for High Performance Numerical Simulation, Beijing 100088, China.

³School of Water Resources and Environmental Science, China University of Geosciences (Beijing), Beijing 100083, China.

Abstract: The desire to increase spatial and temporal resolution in modeling groundwater system has led to the requirement for intensive computational ability and large memory space. In the course of satisfying such requirement, parallel computing has played a core role over the past several decades. This paper reviews the parallel algebraic linear solution methods and the parallel implementation technologies for groundwater simulation. This work is carried out to provide guidance to enable modelers of groundwater systems to make sensible choices when developing solution methods based upon the current state of knowledge in parallel computing.

Keywords: Groundwater flow simulation; Parallel algebraic algorithms; Krylov subspace methods; Preconditioning; GPU computing; Parallel computing framework

Introduction

Groundwater flow simulation in a porous medium often involves solving the diffusion equation which is derived from Darcy's Law and the law of conservation of mass (Harbaugh A W, 2005). As the numerical simulation advances, diffusion equations have become more complex and larger than ever before. Hydrogeologists pay more attention to whether the solution can be accurate enough to be applied in solving actual problems. While the solutions of models are trivial in many instances, circumstances exist that can make the solution more challenging (Miller C T *et al.* 2013). For example, K , the hydraulic conductivity tensor, usually varies in space; in addition, the parameter S_s takes up the specific storage of the porous material over several orders of magnitude and short spatial scales. Although the data typically do not exist to represent this variation precisely, representation of variability in a statistical sense conditioned on known values of

K and evaluated over many realizations can be used to evaluate the effect of heterogeneity and uncertainty, leading to computational challenges, especially as the scale of domain relative to the scale of discretization level becomes larger (Miller C T *et al.* 2013). In such cases, it will be necessary to employ the parallel computing technology and effective solvers. Effective algebraic solution methods and parallel implementations have been widely shown to be useful in solving complex problems. In order to provide readers a better understanding of such methods and implementations, this paper will briefly review and discuss various classic parallel algebraic solution methods and parallel implementations.

1 Algebraic linear solution approaches

1.1 Stationary iterative methods

To solve the diffusion equation (1), which remains the main task in our groundwater numerical simulations, appropriate spatial and temporal discretization methods (such as finite

*Corresponding author. E-mail: jshao@cugb.edu.cn.

difference, finite element and finite volume methods) are applied and a solution approach is

$$\frac{\partial}{\partial x}(K_{xx}\frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(K_{yy}\frac{\partial h}{\partial y}) + \frac{\partial}{\partial z}(K_{zz}\frac{\partial h}{\partial z}) - W = S_s \frac{\partial h}{\partial t} \quad (1)$$

Generally, there are two types of methods in solving the problem, namely the direct solution methods and the iterative solution methods. Though direct methods (for example, some variant of Gaussian elimination methods) can be used for some small 1D or 2D problems, for large-scale 3D applications, the storage requirement for a realistic size problem can simply be prohibitive. In addition, given that simulating actual groundwater-related problems often requires large storage space in nowadays, the most frequently used methods are iterative solution methods.

The classical and the simplest iterative methods are the stationary iterative methods. In these methods, the equation $Ax = b$ is converted to a linear fixed point problem that can be expressed as $x = Mx + b$, where the matrix M is in some sense “close” to the coefficient matrix A . Then the iteration process can be represented as $x_{n+1} = Mx_n + b$. To construct M , a matrix representation of A is used in the classical stationary iterative methods, such as Jacobi, Gauss-Seidel and SOR. From the point of view of the convergence rate, the classical stationary iterative methods perform poorly and the situation can become even worse as one refines the grid. However, these methods can be implemented in a matrix-free way, thus becoming very efficient in terms of memory. Moreover, these methods serve as the basis for some useful preconditioners and some of them such as Jacobi can be easily parallelized.

1.2 Krylov subspace methods

Another popular type of iterative methods nowadays is that of Krylov subspace methods, such as the CG (conjugate gradient) method for positive definite system and GMRES for general system. The essence of these methods is to construct the residuals using some polynomial in the coefficient matrix A , that is $r_n = P_{n-1}(A)r_1$ where r_n is the residual in n th iteration. As shown in Fig. 1, major computational steps in Conjugate gradient-type methods are parallelizable. Vector

required to solve the linear equation in the form $Ax = b$.

updates can be fully parallelized, and the SpMV (sparse matrix-vector product) involves only local communication for most sparse problems. Calculation of norms and inner products takes global operation, but the amount of data involved is small, and the cost usually can be negligible compared to matrix-vector products especially the preconditioner application (Dongarra J J *et al.* 2003).

$$r_0 = f - Ax_0, \quad p_0 = z_0 = P^{-1}r_0$$

for $k = 0, kmax$

$$w_k = Ap_k; \quad a_k = \frac{(z_k, r_k)}{(p_k, w_k)}$$

$$x_{k+1} = x_k + a_k p_k; \quad r_{k+1} = r_k - a_k w_k$$

$$Pz_{k+1} = r_{k+1}; \quad b_{k+1} = \frac{(z_{k+1}, r_{k+1})}{(z_k, r_k)}$$

$$p_{k+1} = z_{k+1} + b_{k+1}p_k$$

check convergence; continue if necessary

endfor

Fig. 1 Computational steps in preconditioned conjugate gradient methods

1.3 Preconditioning

In the past decades, it has been widely recognized that preconditioning is the most critical step in the development of efficient solvers for challenging problems in scientific computation, which is true for Krylov methods in particular, because the memory required and cost per iteration will increase dramatically with the number of iterations. The basic idea of preconditioning is to transform the linear system into another equivalent system by improving the spectral properties of the coefficient matrix, the decisive factor of the convergence rate. For example, MODFLOW PCG2 package provides two kinds of preconditioned CG methods: The MICCG method and the POLCG method. These methods both have great trade off consideration between increased efficiency and increased computer storage. Especially, the MICCG method has better convergence rate and compute efficiency. However, due to the

limited parallelism in incomplete factorization methods, the time-consuming construction of Modified Incomplete Cholesky preconditioner is hardly parallelized, which will largely hamper the parallel performance improvement. The POLCG method exhibits remarkable parallelism and is easy to use; but its OpenMP-based version can merely compete with the serial MICCG method on an eight core workstation (DONG Yan-hui and LI Guo-ming, 2009). Thus, for parallel groundwater flow simulation, the following preconditioners are more preferable.

The simplest but popular preconditioner is Jacobi, simply a diagonal matrix, which is lightweight (almost no construction and application cost at all) and suitable for parallel processing without paying any extra cost. Though Jacobi preconditioned Krylov subspace methods do not always ensure required precision, or even fail to converge for some complex scenarios, they are efficient in solving some easy-to-solve problems, such as groundwater flow in homogeneous or low heterogeneous media, *etc.* (CHENG Tang-pei *et al.* 2014ab).

Another popular preconditioner is the multigrid preconditioner. The essence behind the multigrid preconditioner is to eliminate both high and low frequency errors respectively by using a hierarchy of meshes. In other words, a simple method such as SOR is used on the coarse grid to reduce high frequency errors, while restriction to the remaining grid points is applied to correct the lower frequency errors through the recursive application of this process (Saad Y, 2003). Classical multigrid preconditioner refers to the GMG (Geometry Multigrid) preconditioner, which is applied based on the problem geometry. The GMG preconditioner is optimal but it is problem dependent. In contrast, from a point of view of library software, a less optimal AMG (Algebraic Multigrid) preconditioner

is easier to construct and apply since it is problem independent (which only requires the coefficient matrix as input). Both GMG and AMG methods are efficient in improving the convergence rate of Krylov subspace method and can be parallelized through distributing each level of mesh hierarchy over the processors. However, the construction of these methods may not be trivial for parallelization, and there is often a trade off between the improved convergence rate and the enhanced parallelism.

2 Computational environment and parallel implementations

2.1 Message-passing programming

Over the past several decades, the MPI (Message Passing Interface) has been served as a widely accepted library for parallel groundwater simulation (ZHANG K N *et al.* 2003; Nakajima K, 2011), and it can be executed on both distributed and shared-memory machines. MPI provides a message programming model where processes communicate and coordinate with each other are carried out through using the library functions to send and receive messages. It can scale well to multiple nodes with advantages such as the user's complete control over data distribution and process synchronization, permitting the optimization of data locality and workflow. However, MPI has shortcomings too. It takes tremendous effort to develop and debug, and its implementation always comes with high latency, low bandwidth and difficulty in achieving load balancing. Fig. 2 briefly describes the parallel implementation of SpMV using MPI.

```

join the local parts of vector x by MPI_Allgatherv(...);
for i := 0 → n
  for j := row_start[i] → row_start[i+1]
    Ax[i] += a[j] · x[col[j]];
  endfor
endfor

```

Fig. 2 Parallel implementation of matrix-vector multiplication using MPI

2.2 Shared memory programming

For shared memory programming, OpenMP is an industry standard and has been adopted in the groundwater simulation community for years (DONG Yan-hui and LI Guo-ming, 2009; Hwang H T *et al.* 2014). OpenMP provides a fork-and-join execution model in which the program begins execution as a single thread. The thread will run sequentially until an OpenMP pragma directive for the parallel region is found. And then a group of threads will be generated through a fork operation and will be incorporated at the end of the region. Most of the times, an existing sequential program can be easily parallelized by placing directives around time consuming loops which are free of data dependences, leaving the most part of source code unchanged. Compared with MPI, using shared memory system resource in OpenMP can achieve low latency and high bandwidth. What's more, it adds fine granularity and allows increased and dynamic load balancing, which may lead to performance enhancement. With regard to the disadvantage of the OpenMP, it is only able to

exploit fine-grain parallelism but hard to optimize the memory access, which may result in performance losses.

2.3 Distributed shared memory programming

The hybrid paradigm that combines message-passing and shared-memory parallelism has become a leading trend in software for clusters in the past few years in terms of subsurface simulation (Tang G P *et al.* 2010). Such trend clearly reflects the desire to take advantage of the two programming models. As illustrated in Fig. 3, the paradigm is usually based on a hierarchical model, making it possible to exploit large and medium grain parallelism at the MPI level, and fine grain parallelism at the OpenMP level. Hence, the hybrid program is structured as one of MPI processors at high levels, which is enriched with OpenMP directives in exploiting multi-threading features and globally-shared memory. Fig. 4 briefly describes the parallel implementation of SpMV using hybrid MPI/OPENMP.

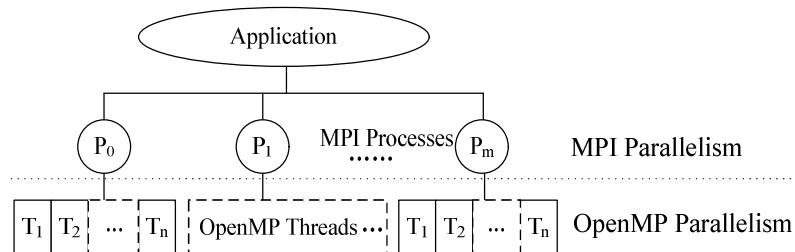


Fig. 3 Hybrid MPI/OPENMP hierarchical model

```

join the local parts of vector  $x$  by MPI_Allgatherv(...);
# pragma omp parallel for private(...) firstprivate(...)
for  $i := 0 \rightarrow n$ 
    recaculating  $col_i$ ;
    recaculating  $a_i$ ;
    for  $j := row\_start[i] \rightarrow row\_start[i+1]$ 
         $Ax[i] += a_i[j] \cdot x[col_i[j]]$ ;
    endfor
endfor

```

Fig. 4 Parallel implementation of matrix-vector multiplication using hybrid MPI/OPENMP

2.4 Programming on GPUs

State-of-the-art Graphics processing units (GPUs) have evolved into highly parallel, multithreaded and core processors, which are attractive hardware platforms for groundwater simulation and other general purpose computing on account of their extremely high floating-point processing performance, huge memory bandwidth and competitive cost (JI X H *et al.* 2012; Hughes J D and White J T, 2013). They have been used to couple two-dimensional overland flow and three-dimensional variably saturated sub-surface flow (LE P V V *et al.* 2015). GPUs offer a tremendous performance improvements and price/performance over the contemporary general-purpose processors. To facilitate parallel programming on GPUs, CUDA (Compute Unified Device Architecture) and OpenCL, programming language has been designed to overcome difficulties in developing application software that transparently scales its parallelism to leverage the increasing number of processor cores. Though remarkable performance improvement can be achieved by using GPU programming, it has high requirement for the understanding of its hierarchy memory structure and extensive performance tuning, which remains an ongoing challenge.

2.5 Programming on parallel computing frameworks

In view of achieving optimal performance using traditional MPI, OPENMP, GPU and their hybrid versions, application developers must have a better understanding of some basic properties of underlying architecture, parallel programming model in addition to physics models. This could be a daunting task, which becomes more difficult in the context of today's ultra-scalable architectures and multidisciplinary as well as the possibly distributed development teams. Challenges mainly arise from the following two increasing complexities. The first is the complexity in the evolving computer architecture. To fully explore the potential of underlying architecture, application developers are required to pay extensive attention to issues such as data-locality and memory access patterns in multilevel memory hierarchy. The

second is the complexity in the evolving applications. To allow newly built algorithms/codes (for example, the linear algebraic solver) to coexist or incorporate the legacy groundwater codes into a multi-physics environment, application developers should spare no effort in the development of "coupler tools" due to the diversity in implementations (such as data structures, interfaces, *etc.*).

For the foregoing reasons, research on the development of coding for groundwater and other related fields under the parallel-computing framework has received considerable attention in recent years, which is evidenced by the newly developed parallel programs such as PFLOTRAN (Hammond G E *et al.* 2014). We have also developed an efficient parallel simulator named JOGFLOW (CHENG Tang-pei *et al.* 2014a) through rebuilding the well-known software MODFLOW on JASMIN (J Adaptive Structured Meshes applications Infrastructure). In general, parallel-computing framework is built on top of traditional parallel programming models with high-level abstraction. To hide parallel-computing details, the framework encapsulates high-performance implementation of data structures, data transfers, load balancing strategies and performance optimizations and provides flexible user interfaces. Through the high-performance computing enabled by data structures and modules reuse, as well as the collaboration between different development teams under the standards of modularization and interface standardization, the complexity of computer architecture can be hidden and the productivity of software development can be substantially enhanced.

3 Conclusions

Despite significant progress in parallel computing over the past few decades, it has not been widely applied in the groundwater simulation community, because of the benefit from the continued increment in CPU clock speed (the Moore's Law). But this picture will soon have to change, which can be observed from the fact that the efficiency of serial computing has approached its limit, while the increased efficiency is becoming increasingly dependent on the parallel computing. The algebraic linear solution approach

ches as well as the parallel implementation are the key to the success of parallel groundwater simulation. From our point of view, the parallel Jacobi and multigrid preconditioned Krylov subspace methods, together with their variants would perform efficiently for the majority of groundwater simulations. With regard to the complex problems which require multi-physics coupling (such as reactive transport), physics-based preconditioner for Krylov subspace methods (such as JFNK) is a new trend and remains as an open research issue (Knoll D A and Keyes D E, 2004). Meanwhile, as the groundwater application and the underlying computer architecture become more complicated, more newly developed parallel groundwater codes will be dependent on high-level languages and parallel computing frameworks in consideration of parallel efficiency and productivity of software development.

Acknowledgements

This research is supported by the National Basic Research Program (973 Program) of China under Grant No. 2010CB428804 and 2011CB309702.

References

- CHENG Tang-pei, MO Ze-yao, SHAO Jing-li. 2014a. Accelerating groundwater flow simulation in MODFLOW using JASMIN- Based parallel computing. *Groundwater*, 52(2): 194-205.
- CHENG Tang-pei, SHAO Jing-li, *et al.* 2014b. Parallel simulation of groundwater flow in the North China Plain. *Journal of Earth Science*, 25(6): 1059-1066.
- Dongarra J J, Foster I, *et al.* 2003. *Sourcebook of parallel computing*. San Francisco: Morgan Kaufmann Publishers.
- DONG Yan-hui, LI Guo-ming. 2009. A parallel PCG solver for MODFLOW. *Groundwater*. 47(6): 845-850.
- Harbaugh A W. 2005. MODFLOW-2005, the US geological survey modular ground-water model: The ground-water flow process. Reston, VA, USA: US Department of the Interior, US Geological Survey.
- Hammond G E, Lichtner P C, Mills R T. 2014. Evaluating the performance of parallel subsurface simulators: An illustrative example with PFLOTRAN. *Water Resources Research*, 50(1): 208-228.
- Hughes J D, White J T. 2013. Use of general purpose graphics processing units with MODFLOW. *Groundwater*, 51(6): 833-846.
- Hwang H T, Park Y J, *et al.* 2014. A parallel computational framework to solve flow and transport in integrated surface-subsurface hydrologic systems. *Environmental Modelling & Software*, 61: 39-58.
- JI X H, CHENG T P, *et al.* 2012. CUDA-based solver for large-scale groundwater flow simulation. *Engineering with Computers*, 28(1): 13-19.
- Knoll D A, Keyes D E. 2004. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *Journal of Computational Physics*. 193(2): 357-397.
- Le P V V, Kumar P, *et al.* 2015. GPU-based high-performance computing for integrated surface-sub-surface flow modeling. *Environmental Modelling & Software*, 73: 1-13.
- Miller C T, Dawson C N, *et al.* 2013. Numerical simulation of water resources problems: Models, methods, and trends. *Advances in Water Resources*, (51): 405-437.
- Nakajima K. 2011. Parallel multigrid solvers using openmp/mpi hybrid programming models on multi-core/multi-socket clusters. In: *High Performance Computing for Computational Science-Vecpar 2010*. Berlin: Springer.
- Saad Y. 2003. *Iterative methods for sparse linear systems*. Philadelphia: Society for Industrial and Applied Mathematics.
- TANG G P, D'Azevedo E F, *et al.* 2010. Application of a hybrid MPI/OpenMP approach for parallel groundwater model calibration using multi-core computers. *Computers & Geosciences*, 36(11): 1451- 1460.
- ZHANG K N, WU Y S, Bodvarsson G S. 2003. Parallel computing simulation of fluid flow in the unsaturated zone of Yucca Mountain, Nevada. *Journal of Contaminant Hydrology*, 62-63(2): 381-399.