Development of a Parallel Computational Framework to Solve Flow and Transport in Integrated Surface-Subsurface Hydrologic Systems

by

Hyoun-Tae Hwang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Earth Sciences

Waterloo, Ontario, Canada, 2012

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

HydroGeoSphere (HGS) is a 3D control-volume finite element hydrologic model describing fully-integrated surface-subsurface water flow and solute and thermal energy transport. Because the model solves tightly-coupled highly-nonlinear partial differential equations, often applied at regional and continental scales (for example, to analyze the impact of climate change on water resources), high performance computing (HPC) is essential. The target parallelization includes the composition of the Jacobian matrix for the iterative linearization method and the sparse-matrix solver, preconditioned BiCGSTAB. The Jacobian matrix assembly is parallelized by using a static scheduling scheme with taking account into data racing conditions, which may occur during the matrix construction. The parallelization of the solver is achieved by partitioning the domain into equal-size sub-domains, with an efficient reordering scheme. The computational flow of the BiCGSTAB solver is also modified to reduce the parallelization overhead and to be suitable for parallel architectures. The parallelized model is tested on several benchmark cases that include linear and nonlinear problems involving various domain sizes and degrees of hydrologic complexity. The performance is evaluated in terms of computational robustness and efficiency, using standard scaling performance measures. Simulation profiling results indicate that the efficiency becomes higher for three situations: 1) with an increasing number of nodes/elements in the mesh because the work load per CPU decreases with increasing the number of nodes, which reduces the relative portion of parallel overhead in total computing time., 2) for increasingly nonlinear transient simulations because this makes the coefficient matrix diagonal dominance, and 3) with domains of irregular geometry that increases condition number. These characteristics are promising for the large-scale analysis of water resource problems that involve integrated surface-subsurface flow regimes. Large-scale real-world simulations illustrate the importance of node reordering, which is associated with the process of the domain partitioning. With node reordering, super-scalarable parallel speedup was obtained when compared to a serial simulation performed with natural node ordering. The results indicate that the number of iterations increases as the number of threads increases due to the increased number of elements in the off-diagonal blocks in the coefficient matrix. In terms of the privatization scheme, the parallel efficiency with privatization was higher than that with the shared scheme for most of simulations performed.

Acknowledgements

First and for most, I express my sincere gratitude to two the best supervisors in the world. First, I would like to thank Dr. Ed Sudicky for challenging me intellectually throughout the process. I feel very fortunate to have just been around him all these years observing how he thinks and approaching problems really help me grow intellectually. To my co-supervisor Dr. Young-Jin Park, I would like to thank you for your ideas and practical knowledge and insights for this project, but also especially for giving me confidence to carry this work through. Without the Zen masters, all of this work would never have materialized. I would also like to thank the members of my committee: Drs. Andre Unger, Sorab Panday, Peter Forsyth, and Ulich Mayer for providing a wealth of knowledge, experience, and guidance. Their valuable feedback helped me to improve this thesis in many ways.

I would like to thank the SciNet staff: Drs. Daniel Gruner, Jonathan Dursi, Scott Northrup, Ramses van Zon, who helped me through any computing problem. Computations were performed on the General Purpose Cluster and Tightly Coupled System supercomputers at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto. Many thanks to present and past members of Dr. Sudicky's group: Rob McLaren, who helped me through any coding problem, as well as Dennis Colautti, Jianming Chen, and Jason Davison, who encouraged me, supported me with my thesis work. I would also like to thank Mary McPherson for reviewing tough drafts and providing valuable suggestions for long time. Thanks to all my friends in Earth Science and Korean Graduate Student Association at the University of Waterloo. There are simply too many names to mention, but I am truly grateful to all those who I have had the privilege to meet and share time (soccer and beers) with along the way, my time at the University of Waterloo was a special experience.

This research was supported by grants to Dr. Ed Sudicky and scholarships to myself (the Canadian Chamber of Commerce in Korea, Atomic Energy Canada Limited, Peter Huyakorn (HydroGeoLogic ltd.), and University of Waterloo graduate scholarships). The financial support was essential for me to reach this point in my career.

Last but not the least I express profound appreciation to my mother, father, younger sister, and younger brother for their unconditional love and tremendous support of my education. I give everlasting gratitude to my fiancée, Yoonhwa Oh, for her love, support and patience throughout my education. I am a fortunate man.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

With population growth, land-use change and urbanization, pressure on the capacity of water resources to meet societal needs is expected to increase with rising demands from agricultural, municipal, industrial, and environmental uses. Prediction of the scarcity and need for water resources can be made more reliable by analyzing the distribution and movement of water as well as the quality of water in the atmosphere, surface and subsurface in the hydrologic cycle. For example, watershed hydrology deals with multiple geologic, hydrologic, hydrogeochemical processes and hydrologic models need to take into account a wide spectrum of factors influencing surface-subsurface water resource such as climate, agriculture, land use, ecosystem, and forest managements. Changing climate will have significant impacts on future water resource and thus the analysis of the impacts may require the consideration of even global-scale coupled atmospheric, oceanic and hydrologic circulation models. In order to solve such complicated water resource problems, numerical modelling is essential.

Among many numerical simulators developed for water resource analyses, HydroGeoSphere (HGS) is a physically-based, fully-integrated surface-subsurface flow and solute and thermal energy transport simulator, developed at the University of Waterloo and the University of Laval. The HGS model's capability has been being expanded to include atmospheric inputs, freezing/thawing of pore water, snow accumulation/melting. The model has been successfully applied to clarify the physical and chemical processes associated with overland flow and variably-saturated subsurface flow in local, watershed, and even continental scales. By simultaneously analyzing surface-subsurface water resources with rigorous consideration of the interactions between them, HGS can evaluate flooding/drought scenarios as well as water quality issues over various temporal and spatial scales. Since hydrologic modeling requires the handling of a wide range of highly nonlinear processes in up to a continental scale for hundreds of years, the computational efficiency of the model has become one of the most critical issues for water resource management as well as the handling of the vast amounts of data involved in hydrological modeling.

High performance computing (HPC) uses supercomputers or clusters of computers to solve advanced numerical problems and has become more and more popular in many fields of science and engineering research. Moore's law for the speed of central computing units (CPUs) – it states the

speed doubles approximately every two years for a single CPU– may have been already violated, and thus, super-computing using multiple processors or parallel computing has become a mainstream methodology for HPC. With the rapid advancement of computer hardware technology, even personal computers are now equipped with multiple processors, with many being interconnected through high-speed networks. Although there exist numerous hydrological simulators for which advanced numerical methods developed for serial computation are implemented to solve the relevant partial differential equations, one of the major barriers for the transition from traditional serial to parallel computing may be the lack of understanding by hydrological scientists on computer hardware architectures, which is essential for the development of HPC tools for the simulators. Under this circumstance, one of the most straightforward ways to adapt HPC for hydrological applications is to utilize pre-developed HPC packages or libraries, which typically provide the interface tools for the application models. As a rule of thumb, it should be noted, however, that trade-offs between generalization and performance cannot be avoided. HPC library packages are mainly designed to handle systems of linear equations, and are not therefore optimal for general hydrological models.

The method to implement parallel HPC in a hydrological simulator is closely related to the hardware architecture of the computing facilities used. Two different standard communication protocols or application programming interfaces (APIs) have been developed to provide tools to control the communication among processors in multi-processor computing machines: Message Passing Interface (MPI) is used mainly for distributed memory systems (DMS) where each processor has its own private memory and as such computational tasks can operate only on local data, while OpenMP is typically used for shared memory systems (SMS) where multiple processors have access to a single shared block of memory. Implementation of OpenMP in SMS is relatively simple for multi-threaded (core) computers including personal computers since the processors can communicate data through shared memory. However, in most cases, the number of computing processors is limited in shared memory systems (at most 124 threads even in a high-end computing facility and 16 in most personal computers). On the other hand, DMS can have thousands of computing units, connected with each other through high-speed networks for massively parallel computing; however, the efficiency of MPI implementation can be limited for some problems if they require the communication of large amount of data among processors.

This research is aimed at implementing a new computational paradigm for integrated water resources quantity/quality management and prediction by applying a parallel computing methodology using OpenMP for hydrologic simulation, which can be operated on most personal computers. It is

noted that the implementation is intended to best utilize SMS at this stage but its future expansion to massively parallel computing using DMS or hybrid distributed-shared systems is considered.

1.1 Previous Research

It has been well recognized in the hydrological community that the dynamics of water movement and distribution in the hydrologic cycle can be better characterized by considering multiple physical, chemical, and biological processes over various spatial and temporal scales. Hydrologic simulators developed over the last 20 years, have expanded their simulation capabilities to include the dynamics of water flow in different flow regimes such as in the vadose zone, over the land surface and in streams and the interactions with the atmosphere in addition to groundwater flow in saturated zone. Because of growing problem sizes and increasing nonlinearities in the mathematical decomposition of the physicochemical processes involved, computational efficiency has become an important research issue for the applicability of these complex numerical simulators. As a key to solving such multi-scale nonlinear problems, there has been a significant amount of research aimed towards developing parallel simulators and improving parallel efficiency in the field of computational hydrology.

Distributed parameter numerical methods to find approximate solutions to the relevant partial differential equations (PDEs) such as finite difference (FD), finite element (FE) and finite volume (FV) methods typically involve a spatial discretization procedure within the simulation domain (meshing or gridding) that convert the differential equations into discrete equations in discrete space (for example, the elemental and global matrix assembly in the FE method). For nonlinear discrete systems of equations (derived from nonlinear PDEs), iterative methods such as Picard or Newton procedures are applied to linearize the discrete equations and then this set of linear discrete equations is solved using iterative or direct matrix solvers. The system of discrete equations can be represented as a coefficient matrix multiplied by a vector of the solution variable equal to the right hand side forcing vector. The characteristics of the matrix equation can be different, depending on the numerical scheme (i.e., spatial and temporal weightings, and element shapes), the model type (i.e., groundwater, variably-saturated, and surface flows) and the boundary conditions applied (i.e., specified and flux boundary conditions). For the solution of the matrix equation, different types of matrix solvers can be used depending on the diagonal dominance, structure of sparsity and symmetricity of the coefficient matrix. By repeating the building and solving of the matrix equations

at each time step over the entire simulation period, hydrological simulators thus spend most of the computing time and resources to build and solve the matrix equations.

Most parallel hydrologic simulators have been developed by implementing existing parallel utilities and solvers such as Aztec [1], DBuilder [2], hypre [3], KINSOL [4], PETSc [5]. These solver packages are typically designed to solve linear systems of equations using multiple processors and to provide interface utilities to link application simulators with the solver packages. Hydrologic simulators with a parallel computational capability include PFloTran [6], TOUGH2-MP [7], FEFLOW/MIKE11 [8], OpenGeoSys [9], ParFlow [10] and WASH123D [11], and there have been attempts to demonstrate the stability and scalability of the parallel implementations. Because many of the parallel solvers available have been developed using Message Passing Interfaces (MPI), these hydrologic simulators can be best operated on Distributed Memory Systems (DMS) such as clusters of workstations and massively parallel processors. In the following, hydrologic simulators having a parallel computing capability and making use of parallel library packages will be briefly reviewed.

1.1.1 Parallel Hydrologic Simulators

Among the parallel hydrologic models listed inTable 1.1, FEFLOW/MIKE11, OpenGeoSys, ParFlow, and WASH123D are capable of modelling groundwater flow and surface water flow, while PFloTran, TOUGH2-MV and TMVOC-MV can simulate only subsurface flow. The parallel solver packages implemented in hydrologic simulators are based on either MPI or OpenMP API. For example, a parallel solver package SAMG (an algebraic multigrid solver), developed using the OpenMP library, is implemented [8] in FEFLOW/MIKE11, and thus the model can be efficiently run on the architecture of SMS. OpenGeoSys applies a domain decomposition approach with the Schur-complement-method [9] and an iterative solver, BiCGSTAB(Bi-Conjugate Gradient Stabilized) to solve the linear system of equations [12]. Parallel hydrologic simulators such as ParFlow, PFloTran, TOUGH2-MP [7], and WASH123D apply parallel preconditioner and solver packages such as Azetc and Portable, Extensible Toolkit for Scientific Computation (PETSc).

Table 1.1 Hydrologic simulators with high-performance computing capability

Simulator	Flow model			Parallel solver package
	Saturated	Variably-saturated	Integrated surface-subsurface	
PFloTran	√	√		PETSc[1)]
TOUGH2-MP	√	√		Aztec[2)]
FEFLOW/MIKE11	√	√	√	SAMG[3)]
OpenGeoSys	√	√	√	DDM[4)]
ParFlow	√	√	√	KINSOL/hypre
WASH123D	√	√	√	DBuilder

PETSc[1)] : Portable extensible toolkit for scientific computing (Argonne National Laboratory)
Aztec[2)] : Iterative library (Sandia National Laboratory)
SAMG[3)] : Algebraic multigrid solver package
DDM[4)] : Domain decomposition method

TOUGH2-MP, a massively parallel version of TOUGH2 [13], is a numerical simulator for saturated, variably-saturated and multiphase flow and multicomponent, solute and thermal energy transport [7]. TOUGH2-MP has been widely applied to analyze radioactive waste disposal, CO_2 sequestration, environmental assessment, contaminant remediation and water resource engineering. In terms of the parallel framework implemented, TOUGH2-MP utilizes Azetc, a parallel solver library, which provides various preconditioners and iterative solvers for large sparse linear systems. ParFlow is an integrated surface-subsurface flow model [10, 14] and it can simulate interactions between the land surface and the atmosphere by coupling a surface/subsurface watershed model and the NCAR Weather Research and Forecasting atmospheric model [15]. In ParFlow, a multigrid preconditioned conjugate gradient algorithm is used to solve the linear system of equations [16] and parallel high performance preconditioners, based on hypre [3], and KINSOL [4] are used for the matrix preconditioning and iterative matrix solution steps. WASH123D is a watershed model for simulating integrated surface-subsurface water flow and contaminant transport [11] that employs a parallel algorithm based on domain decomposition. Specifically, the parallel software implemented in WASH123D is a particle tracking kernel and DBuilder [2]. For particle tracking, a parallel particle tracking algorithm operates by partitioning a simulation domain to processors and by tracking the

particles based on their element locations owned by a processor [17]. DBuilder is used for partitioning the domain and solving the linear systems. PFloTran is capable of simulating thermal-hydrologic-chemical processes with multiscale, multiphase and multicomponent reactive flow and transport [18]. Two modules, PFLOW for flow and PTRAN for transport comprise this simulator [6]. The parallel framework of PFloTran is based on PETSc, a parallel library package for solving linear systems. According to [6], PFloTran implements a domain decomposition approach that distributes sub-domains within the entire simulation domain to individual processors and then solves the linear equations in parallel. For solving the system of nonlinear equations, PFLOW uses SNES, a nonlinear solver module in PETSc, and PTRAN utilizes its own Newton solver routine.

1.1.2 Parallel Solver Packages Used for Hydrologic Simulators

Parallel solver packages have been implemented in hydrologic models to solve large, sparse linear systems. Table 1.2 briefly lists the main computational tasks performed by the parallel packages: 1) partitioning a problem domain, 2) generating a preconditioner and 3) solving the matrix equations.

Aztec is an iterative Krylov subspace solver package that performs parallel computing on clusters with MPI API [1]. Aztec provides data transformation tools for constructing distributed sparse unstructured matrices for the parallel sparse matrix-vector multiplication. The libraries are intended to transform a large matrix into several sub-matrices based on partitioned domains owned by processors [19], which reduces data communication costs among processors that can lower the computational overhead, thus boosting computational efficiency. Aztec supports several options for preconditioners and for Krylov iterative matrix solvers.

Table 1.2 Parallel solver packages

Parallel solver package	Domain partitioning	Preconditioner	Solver type		API	
			linear	nonlinear	MPI	OpenMP
Aztec	√	√	√	√	√	
DBuilder	√		√		√	
hypre	√	√	√		√	√
KINSOL			√	√	√	
PETSc	√	√	√	√	√	

DBuilder is a parallel software toolkit developed using the MPI API and performs a series of parallel computations such as the partitioning of a domain, the management of sub-domain data, and the solution of linear systems [2, 11]. During the partitioning process, DBuilder uses subgraph consisting of vertices and edges to optimize the amount of workload for each processor such that the well balanced workload can reduce the need for excessive communications between processors. For integrated flow simulations, a coupler is provided to send and receive data from one flow domain to another (for example, 2D surface flow domain to 3D subsurface domains). DBuilder also supports linear solvers such as BlockSolver95 (Scalable Library Software for the Parallel Solution of Sparse Linear Systems) and pARMS (parallel Algebraic Recursive Multilevel Solvers). If BockSolver 95 is chosen as a solver, two processes are involved with solving the linear systems: passing the coefficient matrix and domain information to DBuilder and solving the matrix equations with an initial solution guess and a right-hand side vector.

The solver package, hypre, is designed to deal with large, sparse linear systems on massively parallel computers and to provide scalable preconditioners [3]. In terms of data mapping of gridded systems, hypre provides four conceptual interfaces: a structured-grid, a semi-structured-grid, a finite-element structure and a linear-algebraic system interface. The information processed through the conceptual interfaces is passed to parallel preconditioners based on the data structures of the linear systems. Additionally, the interfaces support conjugate gradient solvers for symmetric matrices and GMRES (Generalized Minimal RESidual, [20]) for nonsymmetric matrices. The following parallel preconditioners are available in hypre: SMG and PFMG (semicoarsening multigrid solvers and the algebraic multigrid, Boomer AMG), ParaSails (sparse approximate inverse preconditioner), Saad's dual-threshold incomplete factorization algorithm (PILUT), and ILU(k) and ILUT preconditionings (Euclid). Different from other parallel solver packages mentioned earlier, a hybrid parallel computing – a mixed mode of MPI and OpenMP libraries – was implemented in hypre.

KINSOL is a member of a software suite called SUNDIALS (SUite of Nonlinear and DIfferential/Algebraic equation Solvers) consisting of CVODE (for ordinary differential equations), KINSOL (for nonlinear differential equations), and IDA (for differential-algebraic equations) [4]. KINSOL is a parallel nonlinear solver package developed for distributed memory systems and provides three iterative linear solvers based on Krylov subspace methods: GMRES, BiCGSTAB and TFQMR (Transpose-Free Quasi-Minimal Residual) [21]. The main algorithm of KINSOL is Newton-Raphson iteration, which solves nonlinear differential equations. The parallel schemes applied to KINSOLV are vector product, vector summation and vector norm.

PETSc is a parallel linear or nonlinear PDE solver library and offers standard basic computational and communication kernels [5]. In terms of solving linear systems, PETSc provides parallel algorithms for spatial discretization, partitioning algorithms (i.e., degrees of freedom, cells, nodes, etc.), preconditioners (i.e., Jacobi, Block Jacobi, ILU, Additive Schwarz, multigrid, etc.), Krylov subspace methods (i.e., CG, CGS, BiCGSBTAB, GMRES, TFQMR, Richardson, etc.), and nonlinear solvers (i.e., Newton-Raphson, Line search, Trust region, etc.). The parallel frameworks of PETSc are based on MPI so it performs best on the DMS architecture.

1.2 Objectives

The main objective of this research is to apply parallel frameworks to HydroGeoSphere (HGS) and to develop an efficient and flexible parallel interface. It should be noted that the strategies developed here can be applied to other integrated surface/subsurface hydrogeological models employing finite difference or finite element methods.

Specific objectives are as follows:

- To identify computational hot spots within HGS during various types of simulations and to analyze the computational bottle necks within them to develop parallelization techniques for the improvement of the overall computing efficiency.
- To quantify the robustness and scalability of the parallel interface for HGS; saturated flow, solute transport, variably-saturated flow and integrated surface-subsurface flow models will be considered as the test problems. For element types, structured (block elements) and unstructured (prism elements) grids will be considered for the test simulations. All the test simulations will be investigated to explore the robustness and scalability of Parallel HGS (PHGS). Additionally, real-world large scale simulations will be analyzed to evaluate the parallel efficiency as well as its robustness.
- To perform real-world simulations with higher resolutions using high-performance computing; mesh refinement is a common approach to identify the details of physical processes within a certain area. The present study focuses on developing an applicable mesh refinement technique for high-resolution simulation

1.3 Thesis Organization

Chapter 2 provides the parallelization schemes adapted to HydroGeoSphere. We identify computational hot spots and present parallel schemes appropriate to improve parallel efficiencies: (1) a domain partitioning method and (2) a parameter privatization scheme. In Chapter 3, we evaluate the parallel efficiency of Parallel HydroGeoSphere (PHGS) for various types of simulations. The scalabilities of PHGS for saturated flow, saturated flow and solute transport, variably-saturated flow and integrated surface-subsurface flow simulations are presented for the evaluation of parallel efficiency. Large-scale real-world simulations are also presented for investigating the parallel efficiency and robustness of PHGS. Chapter 4 introduces a mesh refinement and efficient computing scheme. The scheme for high resolution simulation is applied to a real-world large scale simulation for efficiency evaluation. Chapter 5 provides an overall summary and conclusions from this research.

Chapter 2
Parallelization of HydroGeoSphere

2.1 Introduction

For many hydrological applications, there is increasing need to simulate multiple physical and chemical processes over large spatial scales and time frames in three dimensions. For example, in order to meet these needs, the integrated surface-subsurface hydrologic simulator HydroGeoSphere used in this study has been enhanced to include thermohaline flow and transport in the subsurface, contaminant migration and thermal energy transport in the surface-subsurface regimes, land surface processes such as evapotranspiration and atmospheric coupling. With many complex interacting processes being involved at large spatial and temporal scales, computing costs dramatically increase and it is now critical to improve numerical efficiency. Thus, parallel computing with multiple processors is a logical next step to meet the challenges involved when dealing with complex hydrological problems. Ideally the efficiency of the simulator can be enhanced by a factor directly related to the number of computing units available. However, there can be significant barriers to efficiently parallelize hydrologic simulators: there are floating point operations that need to be performed sequentially (dependency) and the cost of managing multiple processors (parallel overhead) can be high. Thus, the optimal efficiency of parallel computing can be achieved by minimizing the data dependency in the numerical method as well as reducing the parallel overhead.

Distributed-parameter hydrologic simulators generally require the repeated building of the linear matrix equations arising from the discretization procedure. Fortunately, building systems of linear equations consists of many independent operations. For example, a finite difference formulation specifies the connections among a set of neighbouring nodes, which is independent from another set of nodes. Thus, the assembly of the coefficient matrices (whose elements represent the coefficients of a linear system of equations) is perfectly parallelizable and straightforward to implement. There are numerous methods available to solve the system of matrix equations. For some simple solution methods such as Gauss-Seidel and Jacobi iterative methods, data dependency is extremely low, but for more advanced solution schemes such as SIP (strongly implicit procedure), PCG (pre-conditioned conjugate gradient) or multi-grid methods, data dependency is typically higher. For example, PCG-like Krylov subspace iterative methods (which is the one used in HGS) sequentially perform many vector-scalar (VS), vector-vector (VV) and matrix-vector (MV) operations

for the inversion of the preconditioned matrix (which is close to the original but computationally easier to invert). Although the VS, VV and MV operations are straightforward to perform in parallel, the inversion of the preconditioned matrix has significant data dependency. When an ILU (incomplete LU) preconditioner is used (as in HGS), data dependency is extremely high for the inversion of the LU matrix (LU solve) even though it is relatively inexpensive compared to the use of a direct solver in a serial computing framework.

Many approaches have been taken to enhance the parallelization efficiency of the LU solve. Specifically, [22] suggested two approaches, blocking (also referred to as multiblocking) and level scheduling, for the parallel implementation of the LU solve and [23] applied a multicoloring scheme to preconditioned generalized conjugate gradient methods. [24] compared the parallel efficiency of the multiblock method (termed multicoloring in his study) with that of a level scheduling (or wavefront) scheme and concluded that the multiblock method performs better than level scheduling of natural ordering with respect to the convergence rate of BiCGSTAB. The idea for the multiblock scheme is similar to that of the domain decomposition method, which assigns various colors to the nodes based on their connections, with the number of colors equaling the number of threads used. Specifically, the colors are selected to form a block consisting of one color such that each block matrix can be solved independently by each thread. There are four major steps for an LU solve with the multiblock method: (1) assign colors to the blocks independently from one another with one color for each, (2) search the nodes that make up the boundary between the colored blocks, (3) solve the interior of an independent block, and then, (4) solve the boundary nodes independently.

Minimizing the number of solver iterations is an important factor for the efficiency of iterative solvers [25]. To increase the convergence rate for iterative solvers, reordering of nodes is necessary to reduce the bandwidth of the coefficient matrix and the number of fill-ins [26-29]. [30] indicated that nested dissection ordering is effective for large sparse systems for parallel computing [31]. Nested dissection ordering has been adopted for iterative solvers [30, 32] as well as direct solvers [31, 33, 34]. According to [35, 36], effective memory localization that stores more data in a closer-to-CPU cache is important to improve the efficiency of matrix solvers because memory access latency can significantly influence the performance of the applications.

The method of parallelism depends on the libraries used for the communication between processors in different hardware architectures. Message Passing Interface (MPI) is an API that is used for distributed memory systems, while OpenMP is used for shared memory systems. For MPI, domain decomposition methods have been extensively adapted to minimize the communication loads

between processors [37, 38]. OpenMP, used in this study, has received much attention due to its implementation simplicity in a high-performance computing environment and also because parallelized simulators using OpenMP can be easily applied to personal computers with multicores.

This section briefly summarizes some general characteristics, the numerical methods used, and computing flow for the integrated hydrologic simulator HydroGeoSphere and then presents a profiling result to document the computing cost as distributed over the various numerical tasks. This is followed by a description of the numerical schemes used to parallelize the HGS model using the OpenMP API. Based on the profiling results, this study specifically focuses on the parallelization of the matrix assembly and the ILU matrix inversion step in the BiCGSTAB solver. Efficient node reordering associated with domain partitioning and memory localization are discussed to minimize the parallel computing overhead.

2.2 HydroGeoSphere: An Integrated Hydrologic Simulator

HydroGeoSphere (HGS) is a three-dimensional control-volume finite element hydrologic simulator describing water flow and solute and thermal energy transport in three-dimensional subsurface domain(s), two-dimensional surface and/or fracture domain(s), and one-dimensional well and tile drain domains. In such multiple interacting flow and transport regimes, integrated surface water and groundwater flow simulations involve various tightly-coupled nonlinear processes such as Manning overland flow, variably-saturated subsurface flow, evapotranspiration and density-dependent flow and transport.

2.2.1 Governing Flow Equations

Integrated surface-subsurface flow simulations can be performed by coupling a variably-saturated flow model to a surface flow model. For the variably-saturated flow model, Richards' equation is applied, but its form is modified to take into account the flux exchange with the surface flow regime. The modified form of Richards' equation describing three-dimensional transient subsurface flow under variably-saturated conditions is given by:

$$\nabla \cdot k_r \mathbf{K} \cdot \nabla h + Q + \Gamma = \frac{\partial}{\partial t}(\theta_s S_w) \tag{2.1}$$

where k_r is the relative permeability of the medium [-], which is a function of the water saturation S_w [-] or the pressure head ψ [L], $\mathbf{K}$ is the hydraulic conductivity tensor [LT^{-1}], h is the total head [L] as $\psi + z$ where z is the elevation [L], θ_s is the saturated water content [-], Q is the volumetric flow rate

per unit volume representing a source (+ve) or a sink (-ve) of the medium [LT^{-1}]. The fluid exchange between the surface-subsurface is represented by Γ. If the storage term in Equation (2.1) is expanded, then the term splits into two storage parts: one related to compressibility effects and the other representing changes in saturation ([39]; [40]):

$$\frac{\partial}{\partial t}(\theta_s S_w) \approx S_w S_s \frac{\partial h}{\partial t} + \theta_s \frac{\partial S_w}{\partial t} \tag{2.2}$$

where S_s is the specific storage coefficient [-]. The governing Equation (2.2) is highly nonlinear due to the nature of the constitutive relations between hydraulic head (h), saturation (S_w), and relative permeability (k_r), which is commonly described by [41] or [42]. The depth-integrated surface flow equation adopted in HydroGeoSphere (HGS) involves 2-D diffusion wave approximation:

$$\frac{\partial d_v}{\partial t} = \frac{\partial}{\partial x}\left(d_o K_{ox} \frac{\partial h_o}{\partial x} \right) + \frac{\partial}{\partial y}\left(d_o K_{oy} \frac{\partial h_o}{\partial y} \right) + d_o \Gamma_o + Q \tag{2.3}$$

where d_v is the volume depth [L], d_o is the depth of flow [L], h_o is the water surface elevation [L], and K_{ox} and K_{oy} are the surface conductances [LT^{-1}] that are changed with the friction slopes of the surface and are approximated by the Manning's equation as

$$K_{ox} = \frac{d_o^{2/3}}{n_x} \frac{1}{\left(\partial h_o / \partial s\right)^{1/2}}; \qquad K_{oy} = \frac{d_o^{2/3}}{n_y} \frac{1}{\left(\partial h_o / \partial s\right)^{1/2}} \tag{2.4}$$

where n_x and n_y are the Manning's roughness coefficients [$L^{-1/3}T$] and s is the direction of maximum surface-water slope [-]. The water depth d_0 and the surface conductances K_{ox} and K_{oy} are complex functions of the dependent variables d_0 or h_0 ($= d_0 + z$), and the complex relationship makes Equation (2.3) nonlinear. Details can be found in [43]. In general, water bodies such as lakes, ponds, and big rivers, have low flow rates so that the surface water gradient ($\partial h_0 / \partial s$) in Equation (2.4) goes to zero. If the gradient approaches to zero, the entries of Jacobian matrix used for Newton-Raphson iteration become infinite. Very large values of Jacobian entries make the model unstable and can cause excessive Newton-Raphson iterations. To avoid the problem, HydroGeoSphere uses a minimum elemental gradient which assigns lower limit on the surface water gradient to 10^{-4}.

Separate surface and subsurface flow models can be combined by explicitly coupling the variably-saturated flow and the surface flow equations, which are Equations (2.1) and (2.3), respectively. In HGS, it is assumed that the two domains are separated by a thin boundary layer. Thus, Γ_o represents a first-order exchange between subsurface and surface domains as follows:

$$d_o\Gamma_o = -\Gamma = (k_r)_{exch} K_{exch} (h - h_o) / l_{exch} \tag{2.5}$$

where $(k_r)_{exch}$ is the relative permeability for fluid exchange [L^2], K_{exch} is the surface/subsurface conductance [LT^{-1}], and l_{exch} is the thickness of the interface layer between surface-subsurface domains [L]. In Equation (2.5), a positive Γ_0 indicates movement from the subsurface to the surface domain. HydroGeoSphere is referred as a fully-integrated globally-implicit model because the surface-subsurface governing Equations (2.1) and (2.3) are solved simultaneously with the coupling provided by Equation (2.5).

2.2.2 Control Volume Finite Element Formulation

The control volume finite element (CVFE) method is based on the concept of combining the finite element and finite difference methods [44]. Specifically, the CVFE method takes advantage of the finite element method, which is computationally efficient and geometrically flexible, and the cell-centered finite difference method, which has continuous interfacial fluxes across the element interfaces. Thus, the fluid mass in each single local element is conserved by the CVFE method. For the discretization of the variably-saturated flow equation, the finite element method uses a weighted residual method combined with a trial solution to solve for unknown nodal values of head ($\hat{h}$) and saturation ($\hat{S}_w$) within a domain V such that

$$\int_V \left[\frac{\partial}{\partial t}\left(\theta_s \hat{S}_w\right) - \nabla \cdot \left(k_r \mathbf{K} \cdot \nabla \hat{h}\right) + Q + \Gamma \right] N_i dV = 0 \tag{2.6}$$

For a given elemental volume v, the control volume method utilizes the same shape functions as weighting functions N_i to approximate the trial solutions $\hat{h}$.

$$\hat{h} = \sum_j N_j h_j \tag{2.7}$$

A fully-implicit temporal approximation results in the following discrete equation for node i

$$\frac{\partial}{\partial t}\left[\theta_s (S_w)_i\right] \int_v N_i dv = \sum_{j \in \eta_i} \lambda_{ij+1/2}^{t+\Delta t} \gamma_{ij} \left(h_j^{t+\Delta t} - h_i^{t+\Delta t}\right) + Q_i + \Gamma_i (h_i, h_{oi}) \tag{2.8}$$

where $\lambda_{ij+1/2}^{t+\Delta t}$ represents the weighted value of relative permeabilities, evaluated at the interface between nodal volumes for nodes i and j, η_i is a set of nodes connected to node i, and γ_{ij} is the influential coefficient between nodes i and j. The influential coefficient takes into account the conductances for neighboring elements connected to a given element and can be defined as

$\gamma_{ij} = \int_v \nabla N_i \cdot k_r \mathbf{K} \cdot \nabla N_j dv$. Thus, it is clear that $\lambda_{ij+\frac{1}{2}}^{t+\Delta t} \gamma_{ij} (h_j^{t+\Delta t} - h_i^{t+\Delta t})$ represents the flow from node *j* to node *i*, and Equation (2.8) represents the mass balance for the control volume associated with node *i* or $v_i = \int_v N_i dv$. To deal with the time derivative in Equation (2.8), the standard finite difference discretization is applied at node i:

$$\frac{\partial}{\partial t}(\theta_s S_w) \approx \frac{v_i (S_w)_i^{t+\Delta t} S_s}{\Delta t}\left(h_i^{t+\Delta t} - h_i^t\right) + \frac{v_i \theta_s}{\Delta t}\left[(S_w)_i^{t+\Delta t} - (S_w)_i^t\right] \tag{2.9}$$

A final form of the discrete equation for Equation (2.6) is as follows:

$$\frac{v_i (S_w)_i^{t+\Delta t} S_s}{\Delta t}\left(h_i^{t+\Delta t} - h_i^t\right) + \frac{v_i \theta_s}{\Delta t}\left[(S_w)_i^{t+\Delta t} - (S_w)_i^t\right] = \sum_{j \in \eta_i} \lambda_{ij+1/2}^{t+\Delta t} \gamma_{ij} \left(h_j^{t+\Delta t} - h_i^{t+\Delta t}\right) + Q_i + \Gamma_i \tag{2.10}$$

Similarly, by using the same approach applied for the variably-saturated flow equation, the surface flow governing equation can be discretized as follows:

$$\frac{a_{oi}}{\Delta t}\left[(d_v)_{oi}^{t+\Delta t} - (d_v)_{oi}^t\right] = \sum_{oj \in \eta_{oi}} (\lambda)_{oioj+1/2}^{t+\Delta t} \gamma_{oioj} \left(h_{oj}^{t+\Delta t} - h_{oi}^{t+\Delta t}\right) + d_{oi}\Gamma_{oi} + Q_{oi} \tag{2.11}$$

In Equations (2.10) and (2.11), the fluid exchange terms $d_{oi}\Gamma_{oi}$ and $-\Gamma_{oi}$ are given as $a_{oi}(k_r)_{exch} K_{exch,oi}(h_i - h_{oi})/l_{exch,oi}$ where the dual nodes oi and i represent surface-subsurface nodes, respectively. In terms of coupling the subsurface and surface domains, two approaches, a common-node and a dual-node scheme, are applicable in HGS to define the exchange flux terms. The common-node scheme is based on the assumption of instantaneous equilibrium between the two domains, while the dual-node approach takes into account the discontinuity of the hydraulic head between two domains. The dual nodes linking the surface and subsurface regimes are, however, assumed to be located at the same physical position, but separated by a thin boundary layer.

2.2.3 Newton-Raphson Linearization

One of the challenges of simulating integrated surface-subsurface flow is to solve the nonlinear discrete equations. Specifically, the discrete mass balance equations for surface-subsurface flow become nonlinear because the terms $(S_w)_i^{t+\Delta t}$ and $\lambda_{ij+\frac{1}{2}}^{t+\Delta t} \gamma_{ij}$ for subsurface flow and d_v and $(\lambda_o)_{ij+\frac{1}{2}}^{t+\Delta t} \gamma_{oij}$ for surface flow are nonlinear functions of the dependent variables h and h_o, respectively. To linearize the discrete equations, the HGS model applies the Newton-Raphson (NR)

iterative method. In the NR procedure for subsurface flow, the residual value at each NR iteration can be defined as:

$$R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L}) = \frac{v_i (S_w)_i^{t+\Delta t,L} S_s}{\Delta t}\left(h_i^{t+\Delta t,L} - h_i^t\right) + \frac{v_i \theta_s}{\Delta t}\left[(S_w)_i^{t+\Delta t,L} - (S_w)_i^t\right] - \sum_{j\in\eta_i} (\lambda_{ij+1/2}^{t+\Delta t}\ \gamma_{ij})^L \left(h_j^{t+\Delta t,L} - h_i^{t+\Delta t,L}\right) - Q_i - \Gamma_i^L(h_i^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L}) \quad (2.12)$$

where R_i^L represents the residual for node *i* at the iteration level *L*. To minimize the residual for a given $h_{i,j\in\eta i}^{t+\Delta t}$, a Taylor expansion technique is used such that

$$R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L} + \Delta h_{i,j\in\eta_i}^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L} + \Delta h_{oi}^{t+\Delta t,L}) =$$
$$R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L}) + \frac{\partial R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L})}{\partial h_{i,j\in\eta_i}^{t+\Delta t,L}} \Delta h_{i,j\in\eta_i}^{t+\Delta t,L} + \frac{\partial R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L})}{\partial h_{oi}^{t+\Delta t,L}} \Delta h_{oi}^{t+\Delta t,L} = 0 \quad (2.13)$$

The NR method generally provides rapid convergence and robust solutions, but is more complex and expensive than other methods such as the Picard iterative scheme [45]. As [46] mentioned, an important reason for these latter two characteristics of the NR method is that the Jacobian matrix is essential for applying Newton-Raphson iteration methods. HGS uses numerical differentiation to construct the Jacobian matrix [47]. With a Jacobian matrix being defined as

$$J_{ij}^L = \partial R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L}) \Big/ \partial h_{i,j\in\eta_i}^{t+\Delta t,L},\ J_{ioi} = \partial R_i^L(h_{i,j\in\eta_i}^{t+\Delta t,L}, h_{oi}^{t+\Delta t,L}) \Big/ \partial h_{oi}^{t+\Delta t,L} \quad (2.14)$$

a set of linearized discrete equations can be obtained to update the dependent variables such that

$$J_{ij}^L \Delta h_{i,j\in\eta_i}^{t+\Delta t,L} + J_{ioi}^L \Delta h_{oi}^{t+\Delta t,L} = -R_i^L \quad (2.15a)$$

$$\Delta h_{i,j\in\eta_i}^{t+\Delta t,L} = h_{i,j\in\eta_i}^{t+\Delta t,L+1} - h_{i,j\in\eta_i}^{t+\Delta t,L} \text{ and } \Delta h_{oi}^{t+\Delta t,L} = h_{oi}^{t+\Delta t,L+1} - h_{oi}^{t+\Delta t,L} \quad (2.15b)$$

A similar procedure can be applied to linearize the surface flow equation, with the residual for a surface node *oi* given as

$$R_{oi}^L(h_{oi,oj\in\eta_{oi}}^{t+\Delta t,L}, h_i^{t+\Delta t,L}) = \frac{a_{oi}}{\Delta t}\left[(d_v)_{oi}^{t+\Delta t,L} - (d_v)_{oi}^t\right] - \sum_{oj\in\eta_{oi}} \left[(\lambda)_{oioj+1/2}^{t+\Delta t}\ \gamma_{oioj}\right]^L \left(h_{oj}^{t+\Delta t,L} - h_{oi}^{t+\Delta t,L}\right) - (d_{oi}\Gamma_{oi})^L - Q_{oi} \quad (2.16)$$

Convergence is assumed to be achieved when either $\max_i \left|R_i^{L+1}\right|$ or $\max_i \left|\Delta h_i^{t+\Delta t,L}\right|$ becomes smaller than pre-specified convergence criteria.

2.2.4 Solver for Linear Systems of Equations

WATSIT (Waterloo Sparse Iterative Matrix Solver), developed by [48], is an iterative sparse matrix solver package operating on the linear algebraic system.

$$Ax = b \tag{2.17}$$

where A is an N × N sparse, non-singular matrix, and b and x are N-length vectors. To obtain solutions for the system, there are three constraints on the matrix A: the matrix A must (1) be real, (2) have non-zero, or non-singular diagonal elements on every row, and (3) be sufficiently positive definite.

During the pre-processing stage, WATSIT (1) scales the main matrix, A , and the right hand side vector, b before proceeding with the standard preconditioning and solve step, and (2) rearranges the matrix rows to place the diagonals at the start of the rows. For the solution process, it (1) analyzes and generates the internal, static data structure for the preconditioner and performs the numeric factorization (preconditioning), and (2) applies one of several available iterative acceleration techniques to solve the matrix system (solution). For preconditioning, the solver package can perform

(1) level-based incomplete factorization, optionally with a red/black element system reduction, or (2) drop tolerance preconditioning to remove elements based on the tolerance assigned by user.

Once the preconditioning phases are complete, the matrix can be solved using various types of iterative schemes such as a PCG (preconditioned conjugate-gradient)-like Krylov subspace acceleration technique (CG, ORTHOMIN, CGS, CGSTAB or GMRES). Among the iterative solvers, BiCGSTAB is one of the fastest solvers and its convergence is much more stable than that of other iterative methods [49]. In addition, BiCGSTAB is known to be robust and it solves linear systems reasonably well for most hydrologic problems [50]. Although BiCGSTAB converges faster than others, one of the issues is the computing time: the computing time of BiCGSTAB increases exponentially as the size of the linear system increases. According to [22, 23, 50], the forward/backward substitutions (also referred to as LU solves or LU) is the main bottleneck of the BiCGSTAB solver.

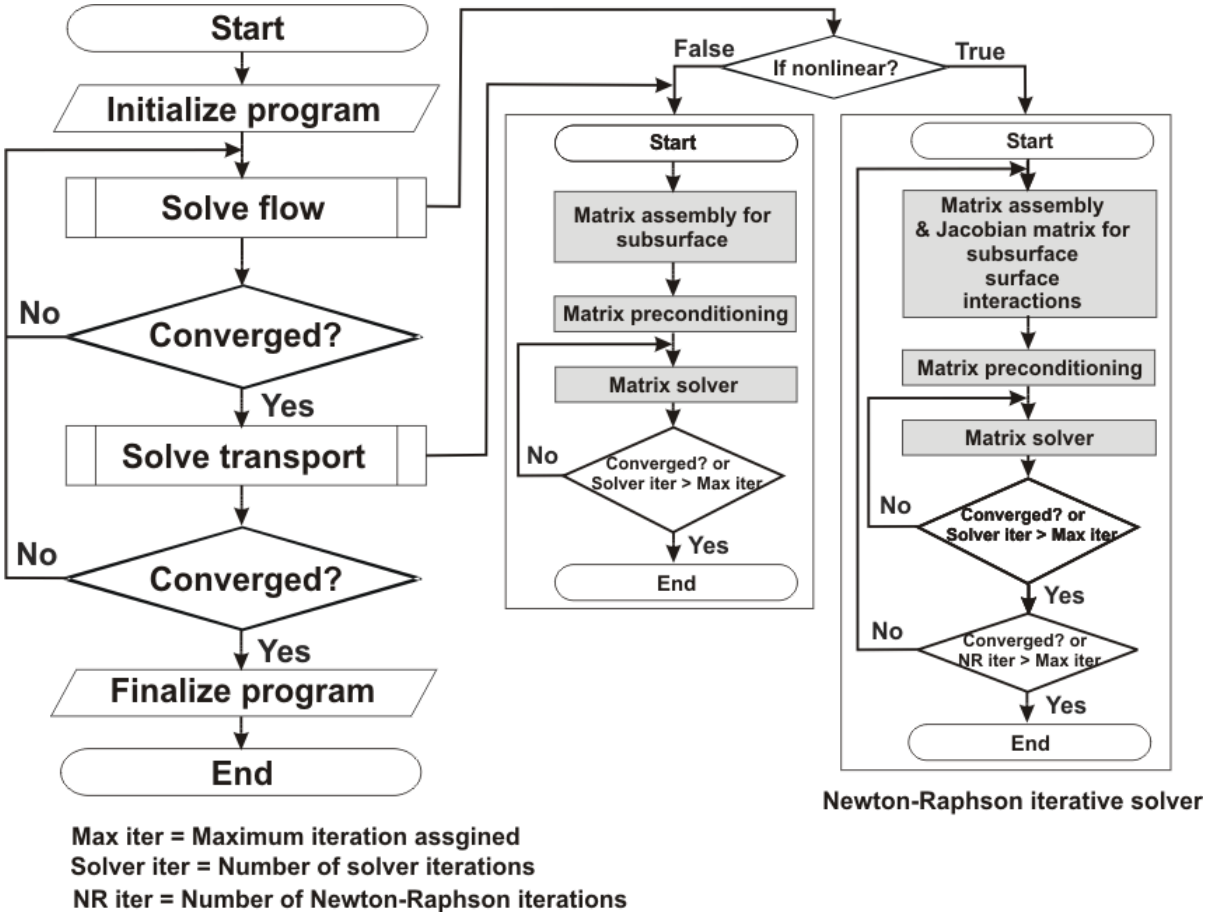

Figure 2.1 Computing flow of HydroGeoSphere simulations

2.2.5 Computing Flow and Parallelization Targets

A schematic of the computing flow within the HydroGeoSphere (HGS) model is illustrated in Figure 2.1. The main tasks can be roughly divided into initialization, simulation time looping, and finalization. For initialization, it reads the mesh, the initial and boundary conditions and initializes the simulation variables and time loops. During time looping, the model repeatedly solves water flow, solute and heat transport at each current time step based on the results from the previous time step until it reaches the final target time (time marching). At each time step, the Jacobian matrix is constructed for the Newton-Raphson iterative procedure for flow, and the solution of the matrix equations are obtained by a preconditioned iterative sparse-matrix solver.

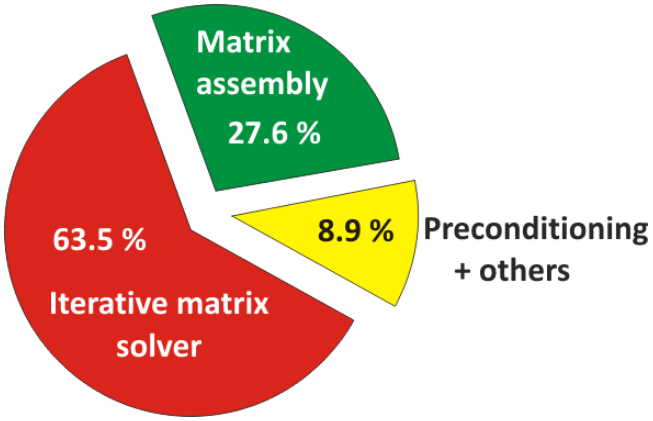

Figure 2.2 A profiling result of the HGS simulation for an example flow problem

In order to build a global Jacobian matrix at each time step, the model sequentially assembles Jacobian matrices for 3-D, 2-D and 1-D domains for subsurface, surface, fractures and wells, respectively. The matrix equation is then passed to the solver to obtain the update of the solution. If the solution at the current simulation time is converged, the solution proceeds to the next time level. If the solution fails to converge, the model takes a modified smaller time step size to repeat the solution procedure. The program finalizes when the final target time is reached.

To identify computational hot spots, a profiling tool called Scalasca [51] was used to analyze the computing cost spent for each computational task when solving saturated flow, solute transport, variably-saturated flow and integrated surface-subsurface flow simulations in domains consisting of 10^4 to 10^7 nodes. The profiling results indicated that more than 90 % of the total computing time is consumed by the matrix assembly and iterative matrix solver (Figure 2.2). Based on the results in Figure 2.2, these two computational tasks (gray boxes in Figure 2.1) were mainly considered for efficiency improvement. Other test problems not discussed here have confirmed that these hot spots are the areas to focus on to improve the computation performance of HGS.

2.3 Parallelization of Matrix Assembly

Although the Newton-Raphson iterative technique generally provides rapid convergence and robust solutions to solve a set of nonlinear discrete equations, [45, 46] and [46] indicate that building Jacobian matrices (numerical differentials) is relatively complex and expensive compared to building a coefficient matrix for the Picard iterative scheme. HydroGeoSphere uses a perturbation strategy for

numerical differentiation to construct the Jacobian matrix [47], and the computing cost for the task is not negligible compared to the total CPU cost.

For the parallelization of the Jacobian (or global) matrix assembly, coarse-grained parallelism is applied. The total number of elements in the domain is divided by a specified number of threads used to pre-determine the groups of elements. Each group of elements is then assigned to each thread for elemental matrix assembly such that an equal amount of workload can be assumed to each thread to build the entire Jacobian (or global) matrix (Figure 2.3). This static scheduling is more efficient than the dynamic, guided and runtime schedulings for this type of parallel numerical tasks [52]. As an example of constructing a Jacobian (or global) matrix with four threads, a domain (20×10×10) consisting of a total of 2541 nodes can be divided into four sub-domains consisting of 500, 500, 500, and 500 elements (Figure 2.3).

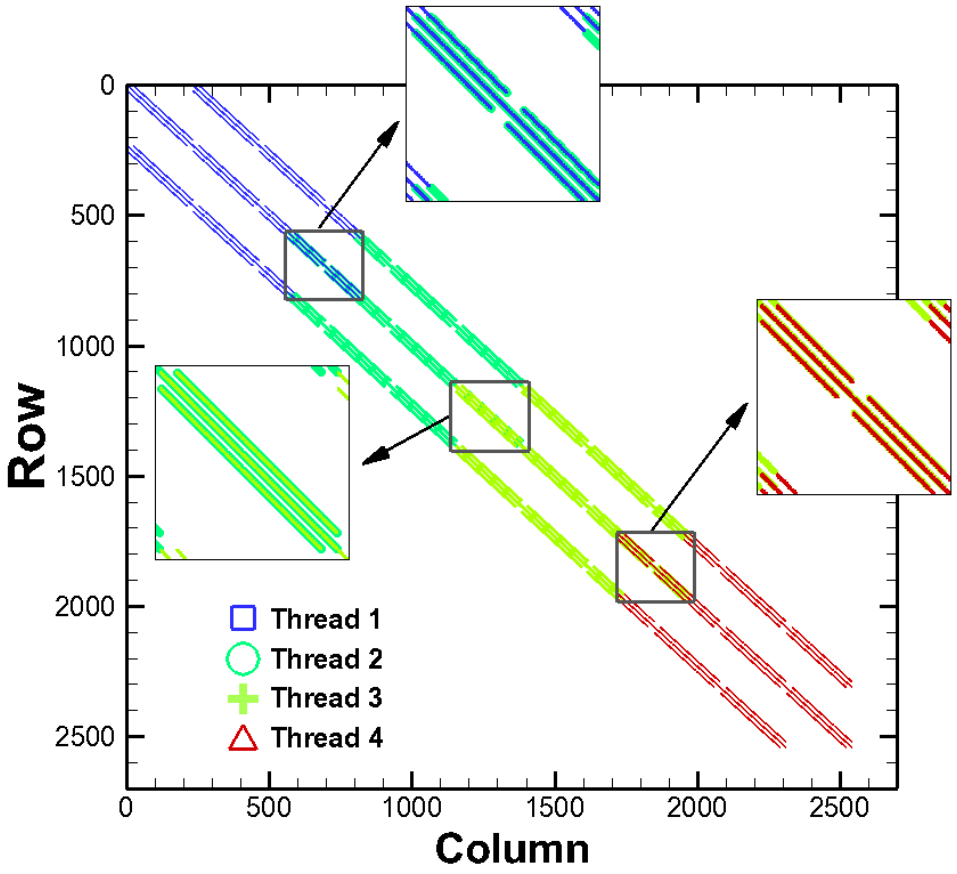

Figure 2.3 Schematic of parallel Jacobian matrix assembly by static scheduling

1) set $\overline{r}_0 = \overline{b} - A\overline{x}_0$ ($\overline{x}_0$ is initial guess)

2) set $\overline{h}_0 = \overline{p}_0 = 0$; $\alpha_0 = \rho_0 = \omega_0 = 1$

3) **For** $i = 1, 2, \ldots,$ until convegence, **do**

4) $\rho_i = (\overline{r}_0, \overline{r}_{i-1})$ ◁ Dot Product

5) $\beta_i = \left(\frac{\rho_i}{\rho_{i-1}}\right)\left(\frac{\alpha_{i-1}}{\omega_{i-1}}\right)$

6) $\overline{p}_i = \overline{r}_{i-1} + \beta_i\left(\overline{p}_{i-1} - \omega_{i-1}\overline{h}_{i-1}\right)$ ◁ Scalr-Vector multiplication

7) solve $\overline{v}_i = (LU)^{-1}\overline{p}_i$ ◁ LU solve

8) compute $\overline{h}_i = A\overline{v}_i$ ◁ Matrix-Vector mltiplication

9) $\alpha_i = \frac{\rho_i}{(\overline{r}_0, \overline{h}_i)}$ ◁ Dot Product

10) $\overline{s}_i = \overline{r}_{i-1} - \alpha_i\overline{h}_i$ ◁ Scalr-Vector multiplication

11) solve $\overline{z}_i = (LU)^{-1}\overline{s}_i$ ◁ LU solve

12) compute $\overline{t}_i = A\overline{z}_i$ ◁ Matrix-Vector mltiplication

13) $\omega_i = \frac{(\overline{t}_i, \overline{s}_i)}{(\overline{t}_i, \overline{t}_i)}$ ◁ 2 Dot Products

14) $\overline{x}_i = \overline{x}_{i-1} + \alpha_i\overline{v}_i + \omega_i\overline{z}_i$ ◁ Scalr-Vector multiplication

15) $\overline{r}_i = \overline{s}_i - \omega_i\overline{t}_i$ ◁ Scalr-Vector multiplication

16) **if** $(\overline{r}_i, \overline{r}_i) \le tol$ **then** ◁ Dot Product

17) method converged

18) **end if**

19) **end for**

Figure 2.4 A pseudocode for the BiCGSTAB iterative solver ($\bar{x}_i$= solution vector of the linear system; $\bar{r}_i$= residual vector of the linear system; A = coefficient matrix; LU = preconditioner)

2.4 Parallelization of Matrix Solver

2.4.1 Preconditioned BiCGSTAB

Figure 2.4 shows the computing flow of a preconditioned BiCGSTAB solver with the ILU preconditioner that consists of five Dot Products (DPs), four Scalar-Vector multiplications (SVs), two forward/backward substitutions (LU solves or LUs), and two Matrix-Vector multiplications (MVs) for each iteration. The number of unknowns (nodes), n, plays a pivotal role to determine the amount

of computing per iteration because the number of the floating-point operations is proportional to n. The total number of floating-point operations required for one iteration of BiCGSTAB (*NFIT*) is the summation of the number of floating point operations for each task times the number of tasks required for each iteration [49]:

$$NFIT = 2nzp + 2nz + 5n + 4n \tag{2.18}$$

where nzp is the number of non-zeros in the preconditioner ILU and nz is the number of non-zeros in the coefficient matrix and n is the number of nodes. Four terms in Equation (2.18) represent the number of floating point operations required for the LU solve, MV, DP and SV, respectively. For example, if a matrix equation is derived using a three-dimensional 7-point finite difference stencil and ILU (0) is used as a preconditioner, nzp is the same as nz (that is $7n$), and thus, the total number of floating point operations per BiCGSTAB iteration is 37 n. Based on this simple example, LU takes about 38 % of the total operations, which is the same as in MV. This example indicates that computing cost may be dominated by the MV and LU solve. According to the profiling result obtained using Scalasca [51], an open-source toolset for analyzing the performance of applications (http://www.scalasca.org/), LU and MV take about 56 % and 37 % of the computing time, respectively. The computing cost of MV and LU is more than ten times compared to the cost used for SV and DP and thus MV and LU components are the computational hot spots for the BiCGSTAB solver. The cost for the MV and LU solve is slightly higher than the result reported in [50] because the test problems applied are different, but confirms that MV is also a computational hot spot in the solver. Modified parallel-BiCGSTAB methods have been suggested to reduce the number of DP operations and to avoid their computational bottlenecks for distributed memory systems (DMS) [53, 54]. Specifically, [54] suggested a new type of BiCGSTAB (termed as Improved BiCGSTAB) without a preconditioner, which can improve computing independence and reduce the number of global communications to one per each iteration.

In the parallel computation for the iterative solver, SV and MV in Figure 2.4 are straightforward for parallelization as they consist of independent computations, while the other components, DP and LU, can only be partly parallelized. Specifically, DP can be performed using the divide-and-conquer approach, which combines all the sub-sums done by each thread so that the synchronization of all threads is necessary to compute scalar parameters such as ρ_i, α_i, and ω_i.

In terms of computing cost, the LU solve is crucial in preconditioned conjugate gradient methods [22, 55] but is one of the most difficult parts to parallelize [55]. To overcome this challenge,

the approach taken here employs two parallelization techniques: (1) multiblocking with the nested dissection method and (2) parameter privatization using a coefficient matrix chopping scheme.

2.4.2 Multiblocking with Coordinate Nested Dissection

The partitioning of a simulation domain into a certain number of smaller blocks is called domain partitioning or multiblocking (Figure 2.5b). The multiblocking scheme is applied for parallelizing the LU solve, which uses the preconditioner for the sparse matrix solve. Figure 2.5 illustrates the multiblocking of a two-dimensional grid system where sub-blocks are represented by different colors with internal and boundary variables being indicated by closed and open circles, respectively. In the multiblocking method, each of the computing processors can perform computational tasks for each of the smaller blocks. During the parallel computation, each processor computes and updates internal and boundary variables in each partitioned sub-domain. Internal and boundary variables in each sub-domain are computed and updated separately by the corresponding processor because they have different dependency characteristics: internal nodes have dependency only among each other and with their own boundaries, while boundary nodes have dependency with their neighbouring blocks.

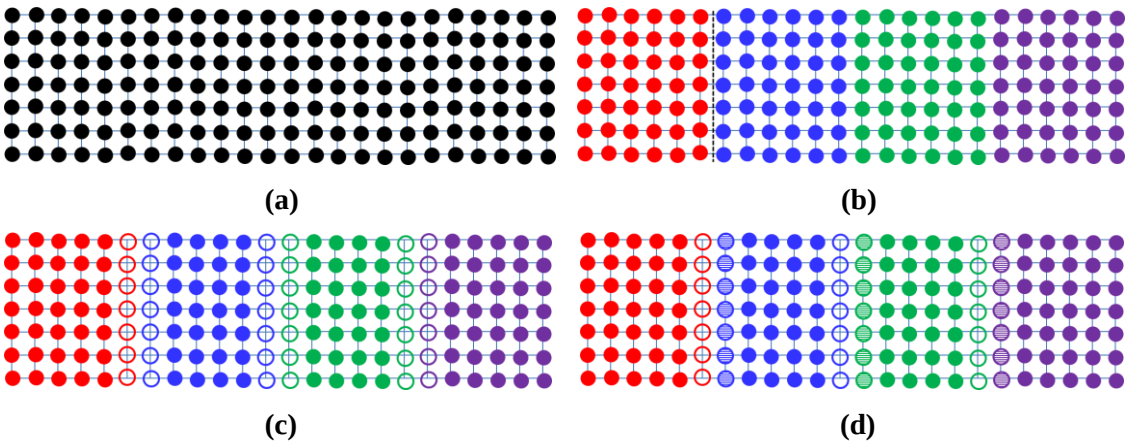

Figure 2.5 Illustration of the multiblocking method. A two-dimensional example grid system in (a) is partitioned into four sub-grid systems in (b). Inside each colored sub-block, boundary nodes (open circles in (c)) have dependency with the nodes of different colors, while the internal nodes have dependency only with the nodes of the same color (closed circles in (c)). Provided the nodes are ordered from left to right, groups of internal nodes have dependency with groups of boundary nodes of higher order (semi-closed circles in (d)).

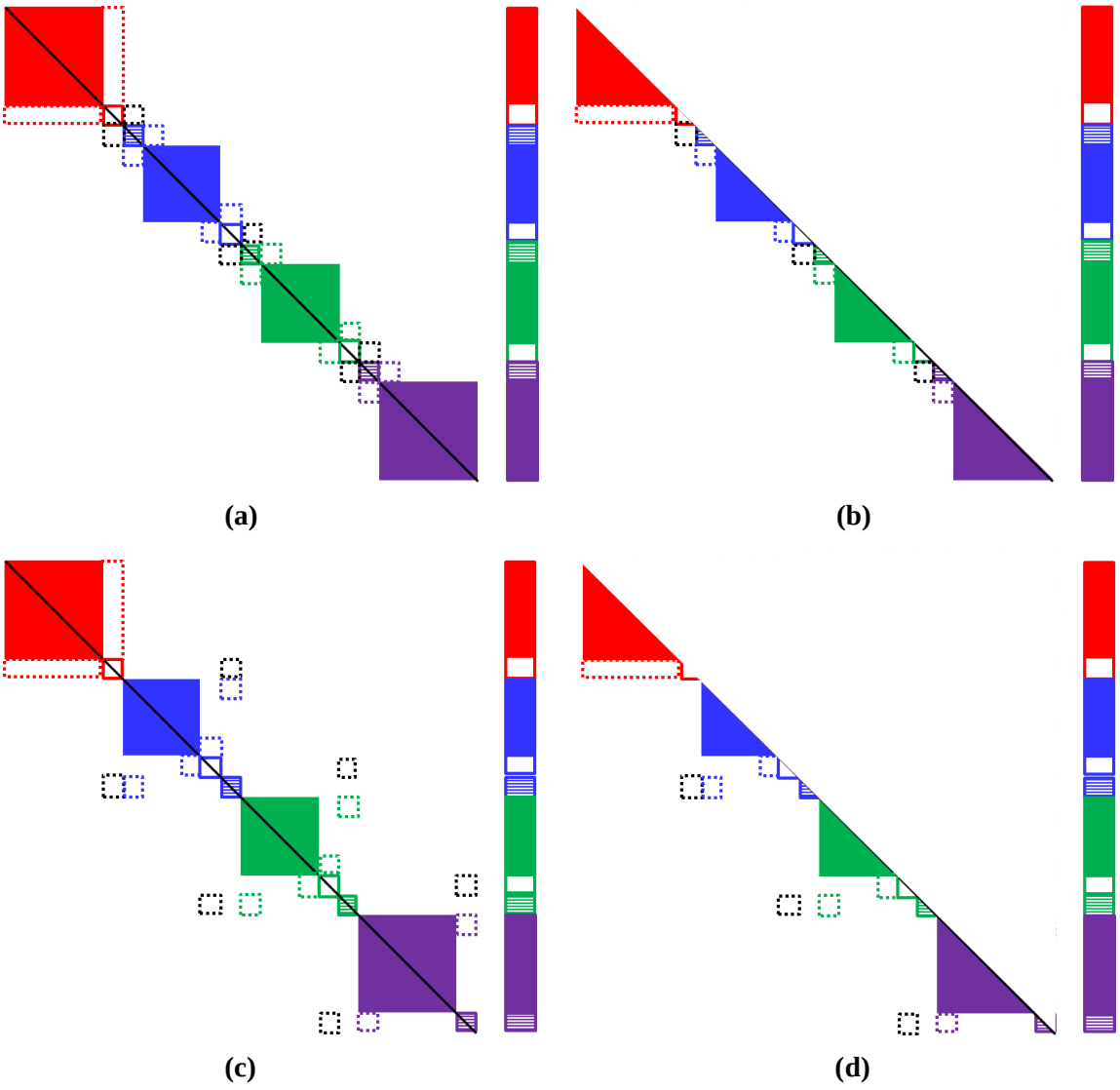

Figure 2.6 (a) shows a matrix system constructed from Figure 2.5d where black dotted element blocks represent the connection between the nodes of different colored blocks. A lower triangular matrix system (b), originated from (a), demonstrates the sequential dependency among blocks. By reordering the blocks as shown in (c), the lower triangular matrix (d) indicates that all the internal sub-blocks are independent from each other.

Figure 2.6a shows a matrix system assembled for the computing grid in Figure 2.5d when the nodes are ordered from left to right (a natural ordering scheme). For forward substitution, lower triangular matrices for the sub-blocks in Figure 2.6b cannot be solved independently due to the dependency among sub-blocks: the red solid block needs to be solved prior to solving the red open block, which also needs to be known to solve the blue semi-solid block. By re-arranging the block order, all the sub-blocks have dependency only on the higher order blocks as shown in Figure 2.6c. Figure 2.6d shows that each of colored blocks has dependency only among each other, except for the semi-solid blocks. Therefore, all the internal blocks and the boundary blocks consisting of open squares can be solved independently and subsequently those boundary variables (semi-solid blocks) can be solved, also independently. Likewise, backward substitution (solution of the upper triangular matrix system) can be performed in an order where the boundary variables of the semi-solid blocks are independently solved first and then the internal node blocks of different colors are solved in parallel.

This study utilizes a coordinate nested dissection (CND) ordering for domain partitioning. According to [25], the coordinate nested dissection scheme provides relatively low quality partitioning compared to the schemes based on the connectivity of the mesh but it is straightforward to implement and is flexible. Using CND, partitioning can be easily modified as necessary to adapt to the characteristics of the physical processes represented in the model. For example, if hydrological processes have the weakest dependency in the vertical direction despite stronger vertical dependency in mesh geometry, partitioning by vertical layers can be easily adapted using the CND scheme.

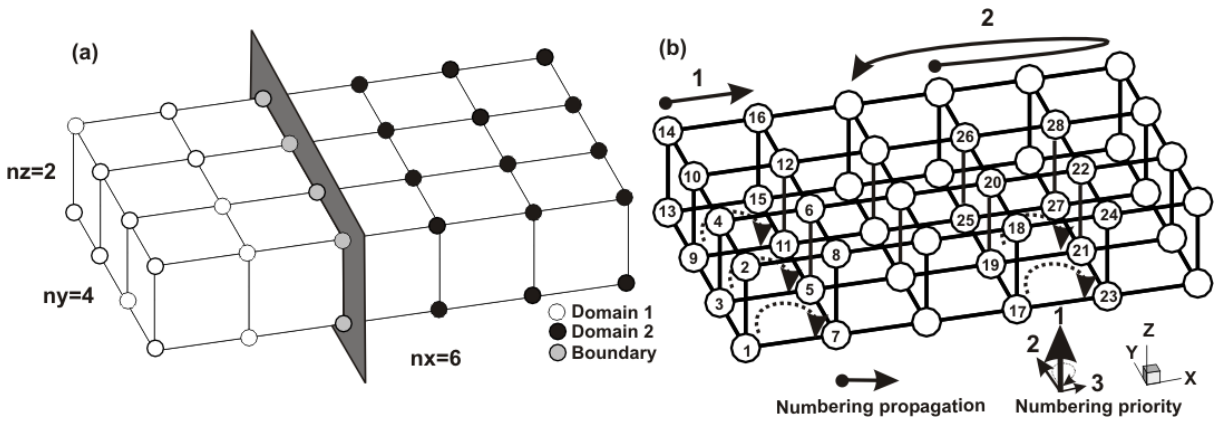

Figure 2.7 Schematic of a) domain partitioning method and b) numbering scheme

Figure 2.7 a shows an example of multiblocking where a simulation domain is partitioned into two sub-blocks (or sub-domains). A group of gray nodes in Figure 2.7a (a separator) breaks the connectivity between the two groups of nodes inside each sub-domain in a way such that all the groups have a similar (or ideally the same) number of nodes. Since the number of nodes along the *x*-axis is the largest, the separator is best chosen to be perpendicular to the *x*-axis to minimize the number of nodes on the separator. Although LU solve can be performed independently for the variables inside sub-domains 1 (open dots) and 2 (closed dots), interprocessor communications are necessary to solve for the variables on the separator, which may reduce parallel efficiency. Except for this communication traffic, LU solve can be performed independently for the sub-blocks. Figure 2.7b shows the ordering process of CND. The ordering priority among the *x*-, *y*-, and *z*-axes is given in the sequence of the *z*-, *y*-, and *x*-axes to minimize the bandwidth of the coefficient matrix. Once the separator and the dimensional ordering priority are defined, the mesh nodes are numbered along the prioritized axes. The ordering method used in this work is unique because when a node (for example, node 1in Figure 2.7b) is numbered, it searches and numbers the neighboring nodes to the node which are not numbered yet (nodes 3, 5, and 7). In each partitioned domain, the main numbering propagation direction is along the *x*-axis, and once all the internal nodes are numbered, the separator is numbered to make a partitioned domain independent from the other surrounding domains.

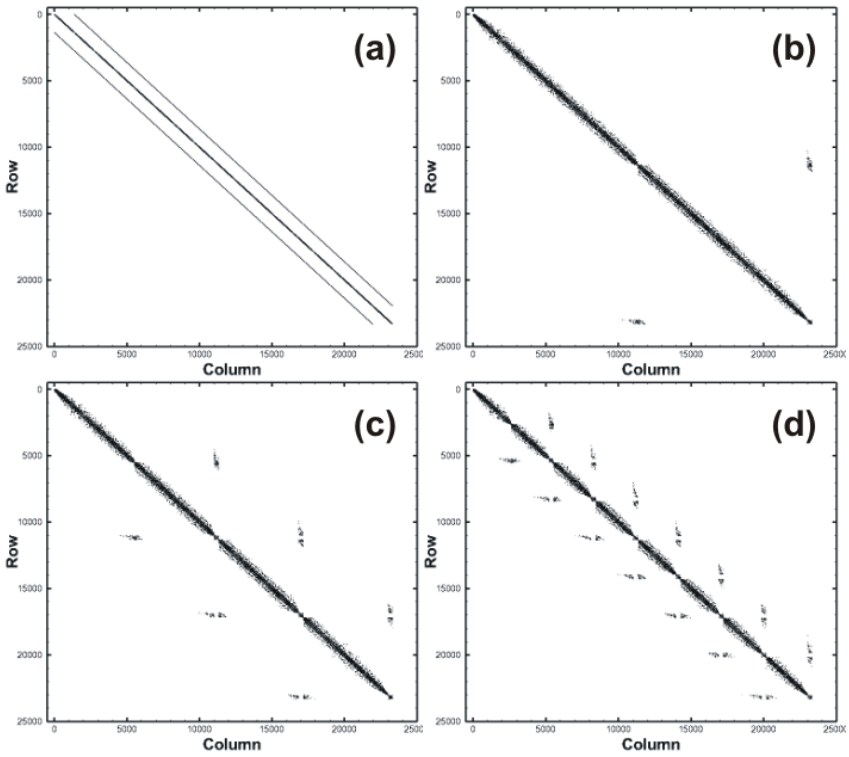

Figure 2.8 Matrix structures of the domain partitioning using a nested dissection scheme: a) original matrix, b) 2 partitioned domains, c) 4 partitioned domains, and d) 8 partitioned domains

Figure 2.8a shows the structure or the nodal connectivity of a coefficient matrix constructed for an example domain consisting of about 25,000 nodes with 50, 50 and 10 grid lines along *x*-, *y*-, and *z*-axes, respectively when a standard finite difference method is used along with a natural ordering scheme. Figures 2.8b, c, and d show the change in the structure of the matrix when the coordinate nested dissection method is applied with 2, 4, and 8 partitioned domains, respectively.

2.4.3 Matrix and Array Compaction for Privatization

In order to minimize the synchronization overhead for OpenMP-based parallel programs where multiple processors access shared variables stored in main memory [28], a parameter privatization scheme was implemented for the WATSIT iterative matrix solver used in HGS. A sub-domain chopping process first needs to be used to apply this scheme where the coefficient matrix and right-hand side and solution vectors are chopped into a pre-determined number of sub-matrices and vectors, and the chopped matrices are then assigned as private variables for the processor operating on them.

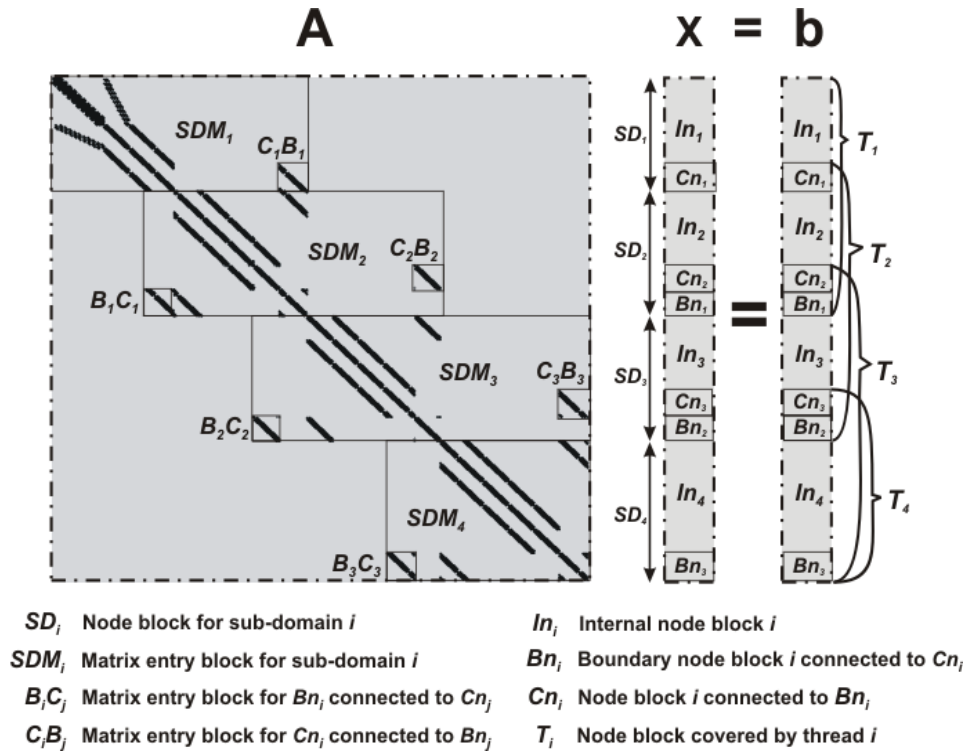

Figure 2.9 Schematic structure of total coefficient matrix for regular parallel scheme

Figure 2.9 illustrates the structure of a typical coefficient matrix (A) and solution and right hand side vectors, x and b. In Figure 2.9, SD_i is the ith sub-domain operated on by thread i; Bn_i is the boundary node block belonging to thread i; Cn_i is the node block of thread i connected to Bn_{i-1}; B_iC_j (or C_iB_j) is the matrix entry block for the connections between Bn_i (or Cn_i) and Cn_j(or Bn_j); and T_i is the size of a node block covered by thread i. The gray area within the dash-dotted lines represents the partitioned structure of the coefficient matrix, generated for a domain size of 2541 nodes (20×10x10). If four threads are used to solve the matrix, the domain can be partitioned into four sub-domains SD_1, SD_2, SD_3, and SD_4 in Figure 2.9. In terms of communication between threads during the forward and backward substitutions or LU solve, the internal node blocks, In_i, need information on a set of nodes connected with the boundary node blocks, Bn_i and Cn_i. For example, Bn_1 is a border between SD_1 and SD_2. To solve the variables in boundary block, Bn_1, the information of Cn_1 belonging to SD_1 is necessary, and thread 2 needs to wait until Cn_1 is solved. Thus, to compute Bn_i, B_iC_j is used in a parallel forward substitution, while C_iB_j is used in a parallel backward substitution. Interestingly, the size of a sub-domain T_i becomes almost doubled because of the

connections to adjacent sub-domains. This doubled data size requires an increase in memory access time leading to parallel overhead costs. A chopping process is suggested here to reduce the overhead cost to access data in memory where the coefficient matrix and vectors are tailored to fit with each sub-domain. This tailoring reduces the data accessing time and increases cache memory efficiency.

Figure 2.10 illustrates the process of stiffness matrix chopping. The coefficient matrix, consisting of four sub-domains, is in the gray area within the dash-dotted lines which is typically assigned as shared parameters for parallel loops. All the variables related to solving the matrix are generally shared among the threads such that each thread can read and write them. The main difference with and without the chopping scheme lies in the size of each sub-domain. With this chopping scheme, the entry blocks for nodes neighboring other sub-domains, such as B_iC_j and C_iB_j, are relocated beside the sub-domain entry blocks. The tailored sub-domains (white dotted box) can now be used only by a single thread as private variables.

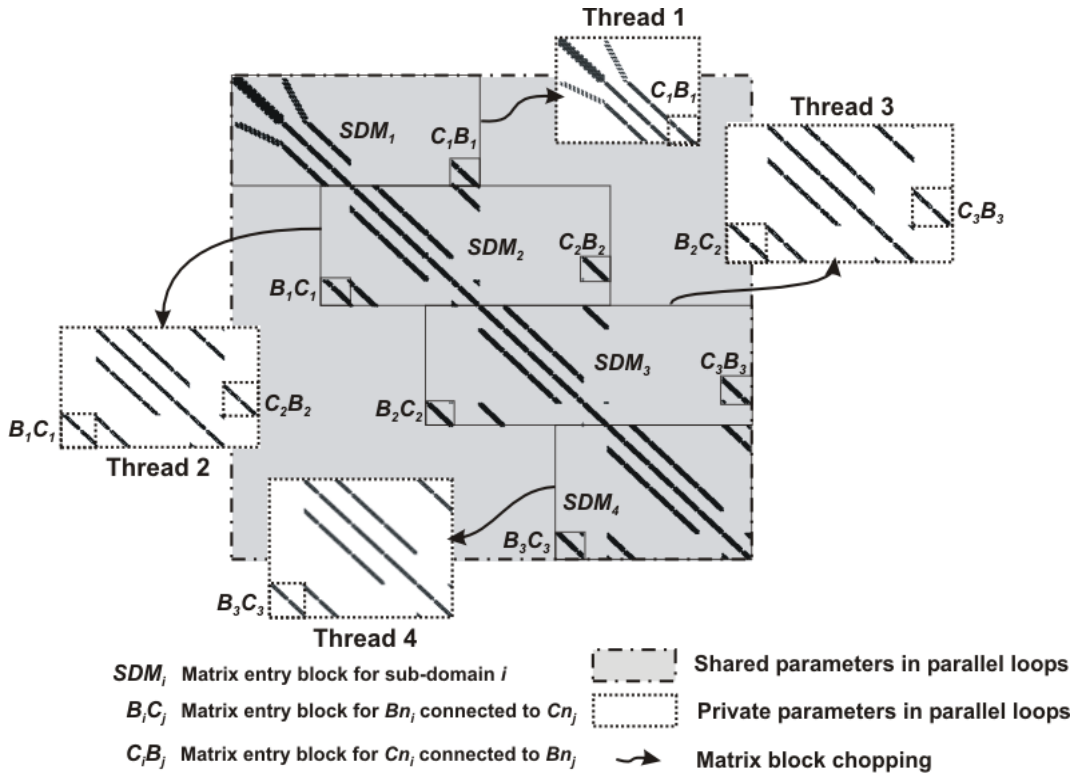

Figure 2.10 Coefficient matrix chopping based on sub-domain

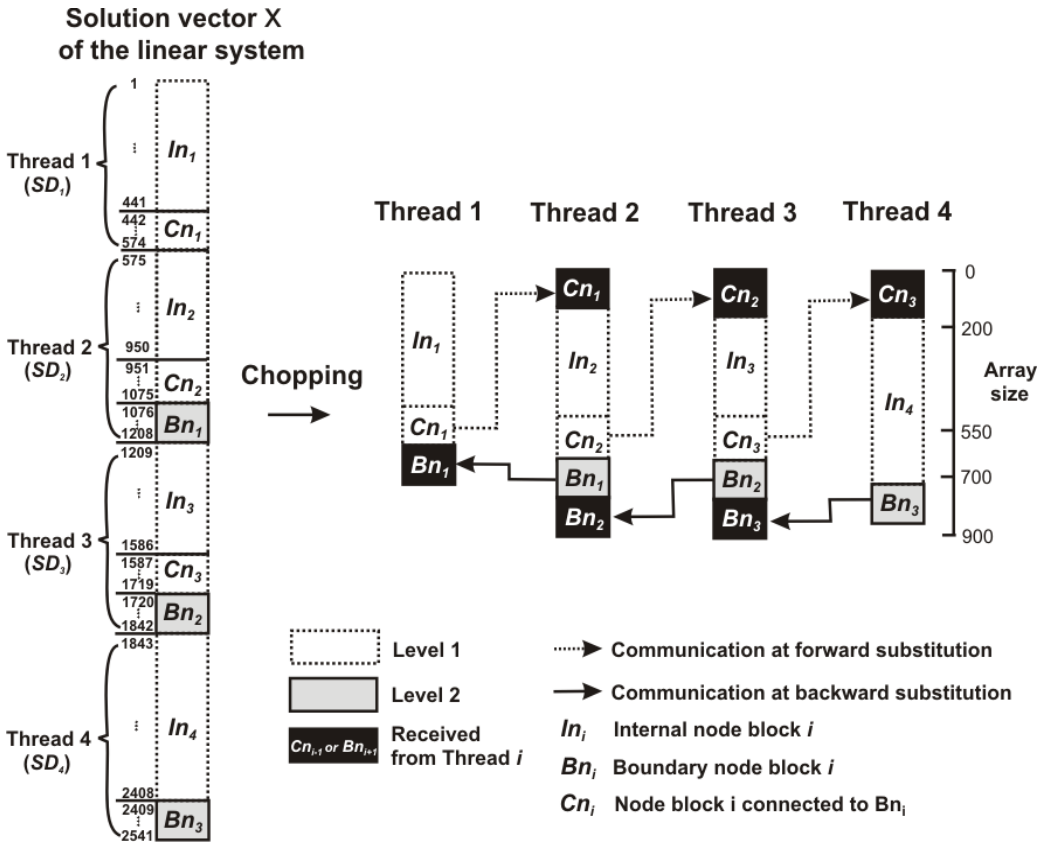

Figure 2.11 Solution vector **X** chopping and communication between threads during the parallel forward and backward substitutions.

Figure 2.11 illustrates the process of chopping for the solution vector and a communication mechanism between the chopped sub-vectors during forward and backward substitutions. SD_i consists of independent block, In_i, boundary block, Bn_i, and the blocks connected to Bn_i, Cn_i after the reordering of nodes. In Figure 2.11, each of SD_1 and SD_4 consists of In, Cn, and Bn, the start and the end of each sub-domain block. SD_2 and SD_3, neighbored by two sub-domains, need additional blocks such as Cn_1 and Bn_2 for SD_2 and Cn_2 and $Bn_3$for SD_3, which are the blocks connected to a previous sub-domain and bordered by the next sub-domain, respectively. During forward substitution, the information on Cn_i, computed directly with In_i, is passed to SD_{i+1} to compute Bn_i (dotted arrows in Figure 2.11). Similarly, during the backward substitution, the information on Bn_i, computed in the forward substitution, is transferred to SD_i to solve Cn_i (solid arrows in Figure 2.11).

With the re-arrangement, forward and backward substitutions are performed in parallel with a static scheduling of loops.

For the specific example shown in Figure 2.11, the coverage for threads 2 and 3 takes a maximum of 1442 addresses before chopping the array, but this reduces to a maximum of 900 addresses after applying the chopping scheme. Reducing the memory access coverage for each thread can significantly improve the parallel efficiency. By applying the privatization scheme to solve the matrix system of equations, the coefficient matrix can be chopped into smaller sub-matrices in which the variables are declared as private. The advantage of using private variables is that no thread needs to access the shared memory and this can reduce parallelization overhead [56] and increase the cache efficiency by reducing the size of the memory allocated to each thread.

Chapter 3
Analysis of Parallel Efficiency

3.1 Introduction

To analyze the parallel efficiency of the hydrologic simulator HydroGeoSphere, parallel simulations were performed using different numbers of processors and the results and performance were compared to those from serial simulations for fully-saturated groundwater flow, solute transport variably-saturated groundwater flow and integrated surface water and groundwater flow. Efficiency of the parallel simulations was analyzed on two different parallel computational facilities provided by the SciNet Consortium [57]: the General Purpose Cluster (GPC) and the Tightly Coupled System (TCS). According to [57], GPC consists of 3864 computing nodes and each of them has two 2.53 GHz quad-core Intel Xeon 5500 (Nehalem) x86-64 processors, with 16 GB RAM per node. TCS has 104 nodes, each with a 16 dual-core 4.7 GHz POWER6 series of processors with 128 GB RAM. The parallelized hydrologic simulator, Parallel HydroGeoSphere (PHGS), was compiled using an Intel® FORTRAN compiler 10.1 for GPC and an IBM xlf compiler for TCS [57].

The number of degrees of freedom involved in a simulation can significantly influence the parallel efficiency. Overall parallel efficiency can become high only when the parallelization overhead is such that the computational load is distributed uniformly over the processors. In general, as more floating point operations are required in a particular simulation, efficiency of the parallel simulator tends to become higher. In this study, the parallel performance is evaluated using meshes with three different spatial resolutions in which the domain was discretized using about 10^5, 10^6 and 10^7 nodes. Simulations with 10^5 and 10^6 nodes are performed using a computing node on GPC with a memory size of 16 GB and TCS was used for the simulations involving 10^7 nodes where the memory size is 128 GB per each computing node. The zero level of fill in, ILU(0), factorization is used for constructing the preconditioner.

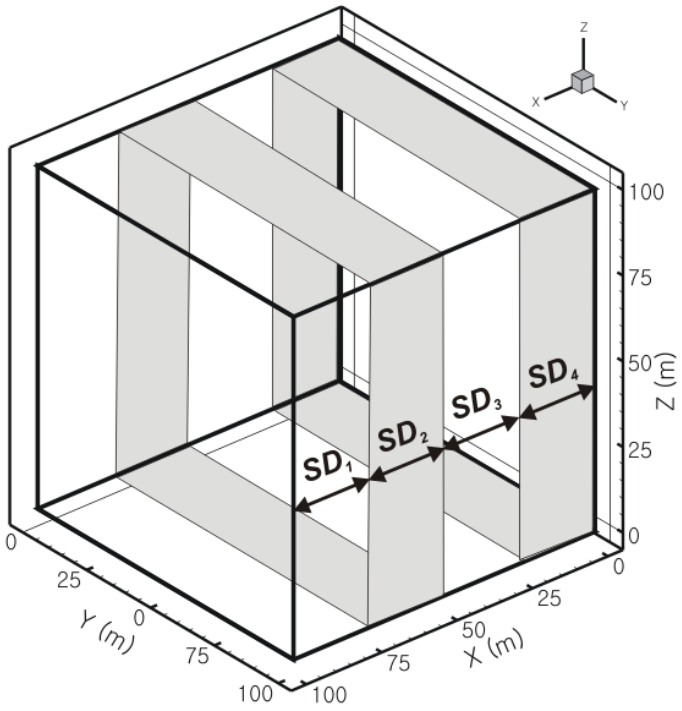

Figure 3.1 Schematic of domain partitioning for four threads

The domain along each axis is discretized uniformly to generate the mesh such that the size of each element decreases as the number of nodes increases. Specifically, the domain is refined along *x*- and *y*-axes to increase the horizontal resolution which typically results in smaller aspect ratio (vertical to horizontal dimension) in large-scale hydrological simulations. It is noted that aligning the direction of all the separators parallel to one another (no pair of separators intersect each other) can reduce communication traffic because all the separators need only to communicate with two blocks. Under this condition, the number of planar separators is $np - 1$, where np is the number of blocks or threads used for the parallel computations. Figure 3.1 shows an example of a domain partitioned for four threads with the coordinate nested dissection scheme. The partitioned sub-domains are equally assigned to the number of threads so that the load balances for each thread are the same.

3.2 Scaling Tests

Two scaling tests are commonly used to evaluate the parallel performance of an application: strong and weak scaling tests. Figure 3.2 illustrates the concept of strong and weak scaling tests when two processors are applied for parallel computing. Strong scaling involves measuring the computing time with an increasing number of processors for a fixed size problem. In the example shown in Figure 3.2 where the total number of nodes for the domain is 125 ($nx = 5, ny = 5$, and $nz = 5$), the domain is divided into two sub-domains each of which has the same size ($nx = 3, ny = 5$, and $nz = 5$) and then the computations in each sub-domain are assigned to two processors (one processor per sub-domain). Computing for the separator can be performed after two sub-domains are simulated in parallel. The ideal computing time obtained with two processors should be one half of that for the serial case involving only one processor. In contrast, weak scaling measures how the computing time varies with an increasing number of processors when each processor has a fixed amount of work load (Figure 3.2). For the case of a weak scaling test with two processors, the number of nodes is assgiend as $(2nx - 1) \times ny \times nz$, and a group of boundary nodes on the y- and z-axes are shared between two processors.In the ideal case, the computing time should remain the same as in the serial computing case.

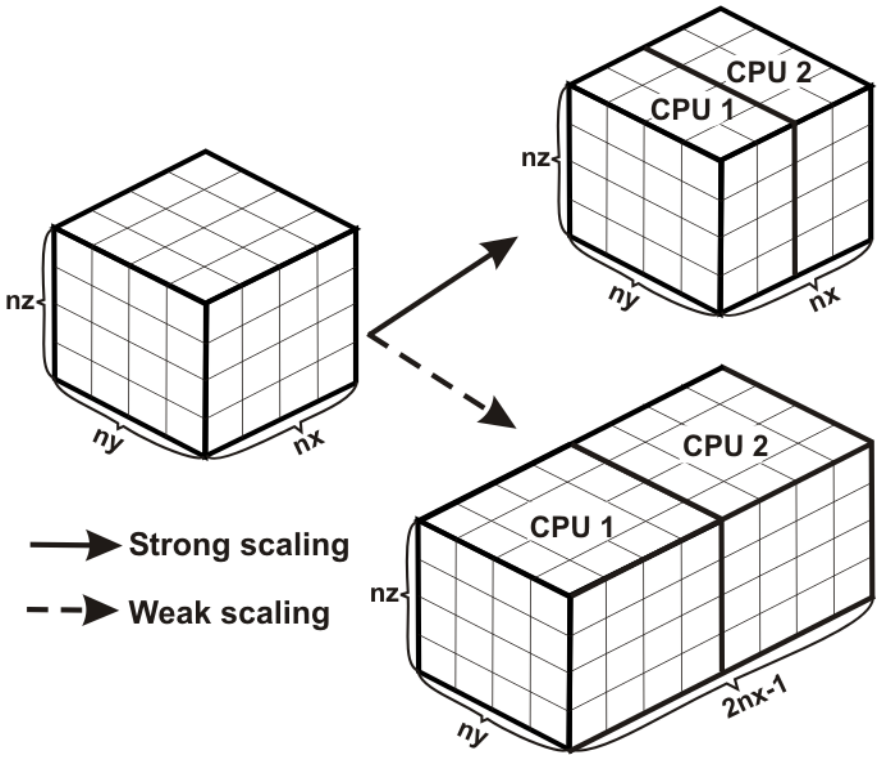

Figure 3.2 Strong and weak scaling tests

For a scaling test, the parallel efficiency of an application can be measured in terms of parallel speedup, S_P, and parallel efficiency, E_p. The parallel speedup for strong scaling, S_P^s, is the ratio between the elapsed wall-clock time with serial computing, T_s, and the elapsed clock time with P processors, T_P, as

$$S_P^s = \frac{T_S}{T_P} \tag{3.1}$$

The speedup, S_P^s, should be greater than 1.0 if the computing time decreases with an increasing number of processors used and ideally $S_P^s \to P$. Therefore, the efficiency for strong scaling, E_P^s, is the ratio between the actual speedup and the ideal speedup [52], given as

$$E_P^s = \frac{S_P^s}{P} \tag{3.2}$$

For a weak scaling test, the parallel efficiency, E_P^w, is the ratio of the elapsed wall clock time obtained with P processors (T_P) and the ideal computing time (T_s: time for serial computing), which is given by

$$E_P^w = \frac{T_P}{T_S} \tag{3.3}$$

The efficiency in weak scaling tests, E_P^w, is generally less than 1.0, but cases when $E_P^w > 1$ are defined as super-linear scalable ones. In this study, the parallel speedup and efficiency are calculated by applying the same computing conditions and algorithms for serial and parallel computing and thus the only difference between them is the number of threads applied.

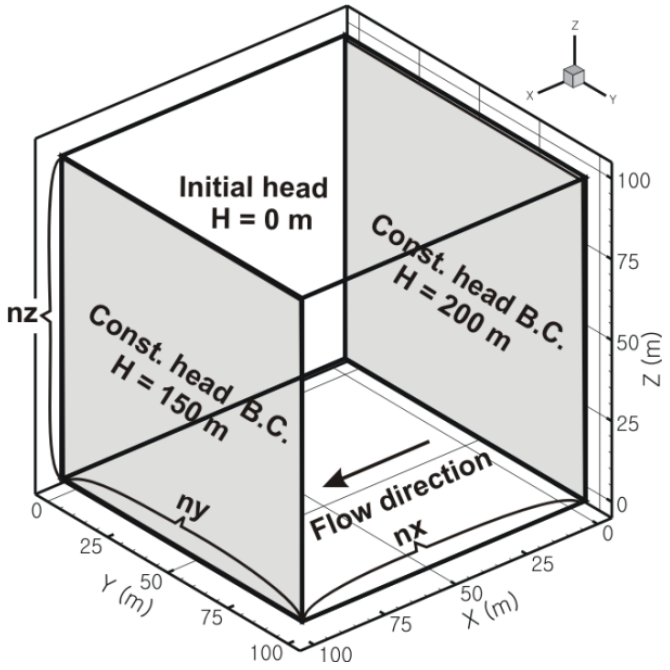

Figure 3.3 Schematic of simulation domain for steady state-saturated flow

3.3 Analysis of Parallel Efficiency for Steady-State Saturated Flow

3.3.1 Simulation Description and Domain Partitioning

To evaluate parallel efficiency involving steady-state saturated groundwater flow, a three-dimensional heterogeneous domain of size 100 m × 100 m × 100 m is used (Figure 3.3). Specified head boundary conditions were applied on the right (x= 0 m) and left (x= 100 m) sides of the domain (H = 200 m at x= 0 m and H = 150 m at x= 100 m), with zero flow being specified on the remaining boundaries (Figure 3.3).

The simulation domain is discretized horizontally with various resolutions: 1 m × 1 m, 0.3 m × 0.3 m, and 0.03 m × 0.3 m and with 10 layers in the vertical direction (Table 3.1). The hydraulic conductivity (K) was assumed to follow a spatially-correlated log-normal distribution and a highly heterogeneous ln (K) conductivity field was generated with a variance of 9.9 and a horizontal and a vertical correlation length equal to10.0 m and 1.0 m, respectively. The highly heterogeneous conductivity field was used to generate computing times meaningful for comparison in serial and parallel modes. Because steady-state saturated flow involves a linear partial differential equation, the

simulations do not require building Jacobian matrices and thus the computing cost for matrix assembly is relatively minor compared to the cost for the solution of the matrix. Thus, the efficiency obtained from these test cases mainly represents the efficiency of the solver. The parallel machines used were GPC and TCS at SciNet. Tests 1 and 2 (the domain consist of about 10^6 nodes maximum), were run on GPC computing nodes having a memory size of 16 GB. Test 3 (with about 10^7 nodes) was performed on TCS, which has a memory size of 128 GB per node.

Table 3.1 Simulation cases for evaluating parallel efficiency (steady state flow simulations)

Simulation No.	Number of nodes (nx × ny × nz)	Heterogeneity		Test machine
		Mean ln($K^{1)}$)	Var ln(K)	
1	100 × 100 × 10	-4.6	9.9	GPC[2)]
2	330 × 330 × 10	-4.6	9.9	GPC
3	3300 × 330 × 10	-4.6	0.0	TCS[3)]

$K^{1)}$: the hydraulic conductivity (m/day).
GPC[2)]: General Purpose Cluster at SciNet/University of Toronto
TCS[3)]: Tightly Coupled System at SciNet/University of Toronto

3.3.2 Consistency of Parallel Simulations

The accuracy and consistency of the solutions obtained using the parallel framework are investigated now. For parallel computing, the node numbering (reordering) is determined by the partitioning of the domain and thus by the number of threads used. Even for the same matrix equation, therefore, the number of iterations for the solver can be different, depending on the number of threads used. The number of solver iterations with reordering is compared to that for serial computing where a natural ordering method is used. The solutions from the simulations with parallel computing are also compared to the serial computing solutions to investigate the consistency.

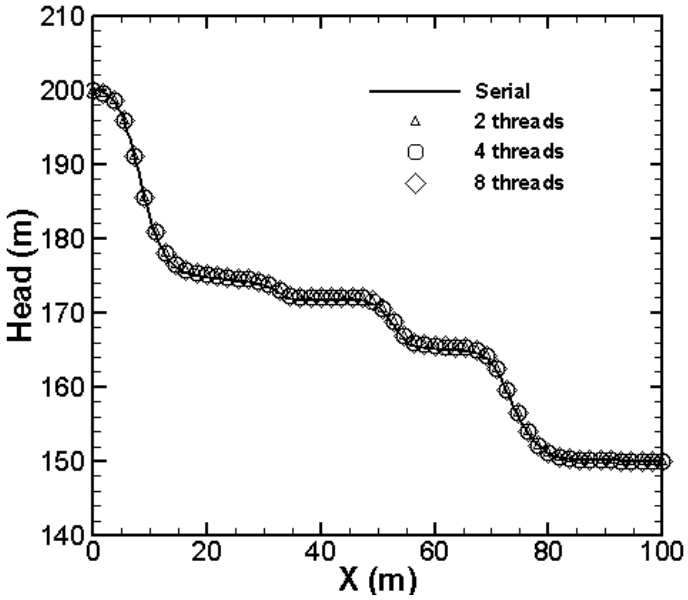

Figure 3.4 Simulation results for serial and parallel computations for saturated flow. The head profile is at y = 50 m and z = 0 m.

Figure 3.4 provides the hydraulic head distribution along the observation line parallel to the *x*-axis at *y* = 50 m and z = 0 m. The head distributions clearly show the effect of heterogeneity in the conductivity field. Comparison of the results from the serial and parallel computations indicates that the maximum difference in head is less than $10^{-7}\ m$ (on the order of round-off error).

The number of solver iterations obtained with natural node ordering is 465 for simulation 1, 1818 for simulation 2, and 1321 for simulation 3 (Table 3.2). Because simulation 3 was performed under the condition of a homogeneous K field, it required less iterations than in simulation 2. For the parallel simulations with node reordering, the number of solver iterations with one thread is similar to that for the serial case, which is used as a reference for estimating the parallel speedup. The results in Table 3.2 indicate that the number of iterations increases with the number of threads applied. The increase in the number of iterations is less than 9 % except for the case of simulation 3 in which 8 threads are used. Simulation 3, with 8 threads, takes a similar number of iterations as that for serial computing with natural ordering: parallel simulations with reordering take about 20% fewer iterations than that with natural ordering. A comparison indicates that the efficient reordering scheme can reduce the number of iterations for saturated flow simulations even with highly heterogeneous conditions.

Table 3.2 Comparison of the number of solver iterations for steady-state flow case 1.

Number of Threads	Simulation 1 (465)[1)]		Simulation 2 (1818)[1)]		Simulation 3 (1321)[1)]	
	Privatized	Shared	Privatized	Shared	Privatized	Shared
1	467	476	1635	1562	693	676
2	454	449	1615	1625	592	647
4	456	477	1693	1745	789	694
8	473	508	1559	1611	553	684
16	NA	NA	NA	NA	863	817

[1)]: Number of iterations in serial computing without reordering

3.3.3 Results of Strong Scaling Tests for Steady-State Saturated Flow

The parallel computation speedup using PHGS was evaluated based on Equation (3.1). The total computing time was calculated by summing the times spent for three major computing tasks: matrix assembly, preconditioning and iterative matrix solution. Because the overall computing time (taken from the beginning to the end of the simulation, which included parts of the code that are not parallelized for pre- and post-processing such as reading input data and writing simulation results) is not indicative of the efficiency of the parallel computation, the time for pre- and post-processing in the simulations was excluded to measure the computing time and efficiency. Simulations 1 and 2 were performed with and without optimization to check the efficiency change when the code is optimized for speed.

In the following sections, the results for the speedup by parallel computing are compared with and without the privatization scheme. To investigate the contributions by the different computing tasks to the parallel speedup, two parallelized components (matrix assembly and matrix solution) were taken into account and their parallel speedups were calculated separately.

3.3.3.1 Results for Saturated Flow Simulation 1

For the case where Parallel HydroGeoSphere (PHGS) was compiled without an optimization option, the parallel evaluations are as follows: Table 3.3 lists the total computing times with a single thread, T_s, and multiple threads, T_P, for the natural ordering and reordering schemes. The T_s values are obtained with both natural and reordering schemes. Specifically, T_s with natural ordering is obtained

from one sub-domain only, while the T_s values with reordering are obtained from 2, 4 and 8 sub-domains and they are used for calculating the parallel efficiency for strong scaling according to Equation (3.1). For the parallel simulations, the T_P values are obtained with the same numbers of threads as the numbers of sub-domains. The T_s value obtained from the serial version of HGS with natural ordering is similar to that obtained from the parallel version with the node reordering method regardless of the use of the privatization scheme. The parallel computing time, T_P, decreases with an increase in the number of partitioned domains. The cases with privatization are consistently faster than those with the shared scheme.

Table 3.3 Total computing times for serial (T_s) and parallel (T_P) jobs with respect to the number of threads used without code optimization for speed for steady-state flow case 1.

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	39[1)]	35 (18)	35 (10)	39 (6)
Shared		35 (21)	38 (16)	40 (10)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

Figure 3.5 shows the parallel speedup and efficiency results for simulation 1 without code optimization. The parallel speedups for the cases with and without privatization become greater with an increasing number of threads used. The maximum speedup using the privatization approach is about 6.1 ($E_8^S = 0.76$) for matrix assembly and 6.7 ($E_8^S = 0.84$) for the matrix solver using 8 threads. Similarly, the maximum speedup without privatization is about 7.3 ($E_8^S = 0.91$) for global assembly and 4.0 ($E_8^S = 0.50$) for the solver. Maximum overall speedup with the privatization scheme is 6.4 ($E_8^S = 0.80$), and about 4.0 ($E_8^S = 0.50$) without privatization when 8 threads are used.

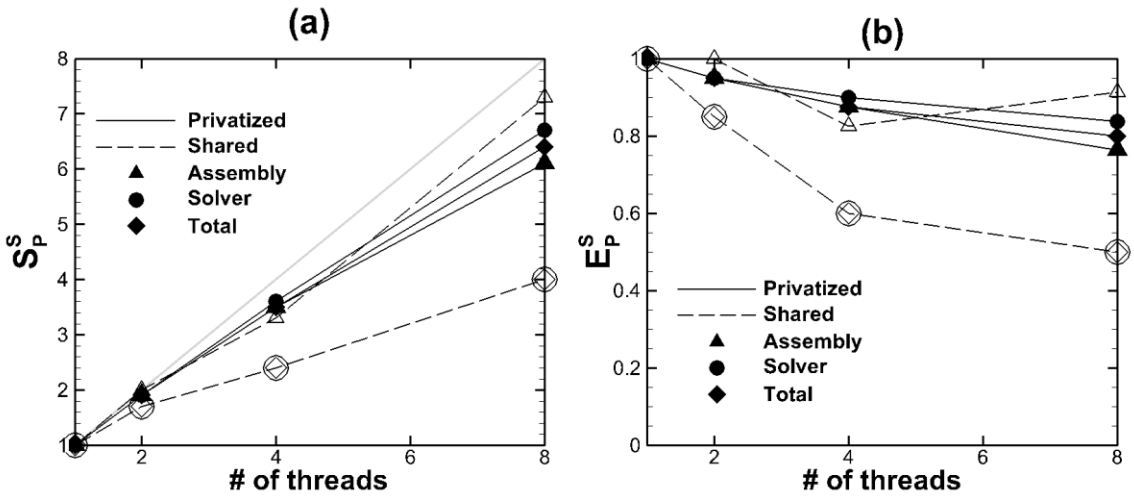

Figure 3.5 Results of speedup and parallel efficiency for steady-state saturated flow case 1 consisting of 10^5 nodes with non-optimized version of parallel HydroGeoSphere: a) parallel speedup, b) parallel efficiency

The results in Figure 3.5b indicate that the parallel efficiency deceases with an increasing number of threads used for the cases with and without privatization. Although parallel efficiency is relatively high for matrix assembly without privatization, the overall efficiency was consistently higher with privatization. By applying the privatization scheme, the parallel efficiency was improved within a range from 10 % up to 30 %.

The absolute computing times for the serial (T_s) and parallel (T_P) computing for case 1 are listed in Table 3.4. The computing time with natural ordering (1 SD) is about 1.3 times more than the mean computing time with reordering (T_s for 2, 4 and 8 SDs). T_s with the privatization scheme is similar to that obtained with the shared scheme.

Table 3.4 Total computing times for serial (T_s) and parallel (T_P) jobs obtained with respect to the number of threads using an optimized version of PHGS for steady-state flow case 1.

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	23[1)]	19 (9)	19 (6)	19 (4)
Shared		19 (11)	19 (8)	16 (6)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

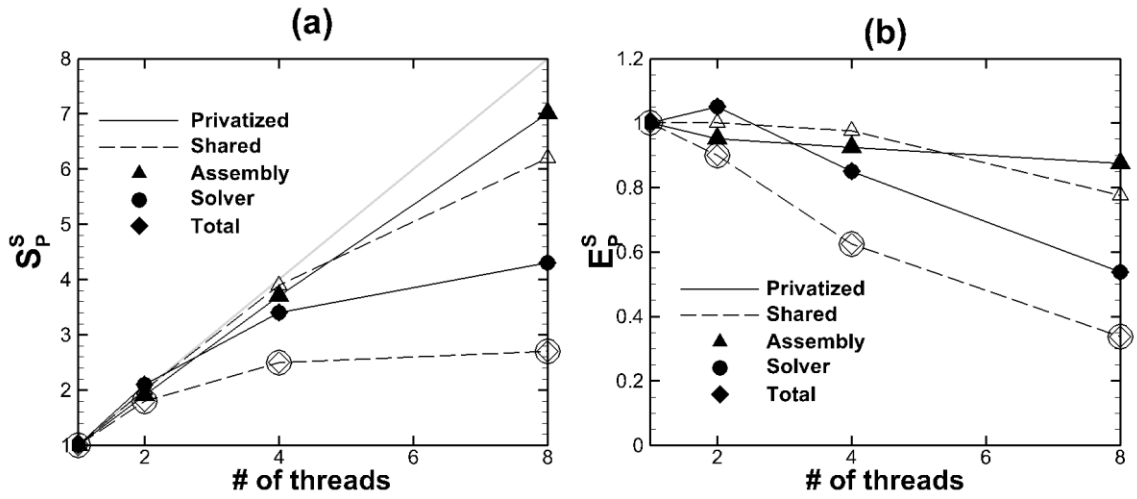

Figure 3.6 Results of speedup and parallel efficiency for steady-state saturated flow case 1 consisting of 10^5 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

The parallel speedup (S_P^S) and efficiency (E_P^S) obtained with the optimized code are compared with and without the privatization scheme in Figure 3.6. Maximum speedup obtained with privatization is 7.0 ($E_8^S = 0.88$) for matrix assembly, 4.3 ($E_8^S = 0.53$) for matrix solution, and 4.3 ($E_8^S = 0.53$) overall, while it is 6.2 ($E_8^S = 0.78$) for matrix assembly, 2.7 ($E_8^S = 0.33$) for matrix solution, and 2.7 ($E_8^S = 0.33$) overall with the shared scheme. The parallel speedup for both schemes becomes larger as the number of threads increases (Figure 3.6a). However, S_8^S value from the shared scheme is similar to the S_4^S value, which results in a decrease of E_8^S to less than 0.40 (Figure 3.6b). By applying the privatization scheme, the parallel efficiency for the solver is improved for all the tested numbers of threads.

3.3.3.2 Results for Saturated Flow Simulation 2

With Parallel HydroGeoSphere (PHGS) compiled without an optimization option, the parallel efficiency is now evaluated for steady-state flow simulation 2. Table 3.5 lists the overall serial (T_s) and parallel (T_P) simulation times for simulation 2 with natural node ordering as well as with node reordering for the different numbers of partitioned domains. The computing time with natural ordering is relatively large compared to those for the reordering cases under serial computing: the reordering scheme improves the computing speed by 26 % compared to the case with natural ordering. T_s values for simulation 2 are about 30 times larger compared to those for simulation 1. Simulation conditions such as different heterogeneous fields between simulations 1 and 2 and, thus the increase in T_s, are not caused solely by the increased number of nodes. T_s with the privatization scheme is larger but the difference is less than 2 %.

Figure 3.7 illustrates that results for parallel speedup and efficiency with and without privatization. The parallel speedup for both cases linearly increases with the increasing number of threads used and the increasing rate with privatization is high. The maximum parallel speedup for privatization is about 7.2 ($E_8^S = 0.90$) for global assembly and 7.4 ($E_8^S = 0.82$) for matrix solution. Similarly, the maximum speedup without privatization is about 7.2 ($E_8^S = 0.90$) for global assembly and 3.1 ($E_8^S = 0.39$) for matrix solution. Overall the maximum speedup with privatization is 7.3 ($E_8^S = 0.92$), and it is 3.1 ($E_8^S = 0.39$) without privatization.

Table 3.5 Total computing times for serial (T_s) and parallel (T_P) jobs obtained with respect to the number of threads using a non-optimized version of PHGS for steady-state flow case 2.

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	1817[1)]	1418 (648)	1257 (358)	1365 (188)
Shared		1354 (860)	1357 (633)	1298 (417)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

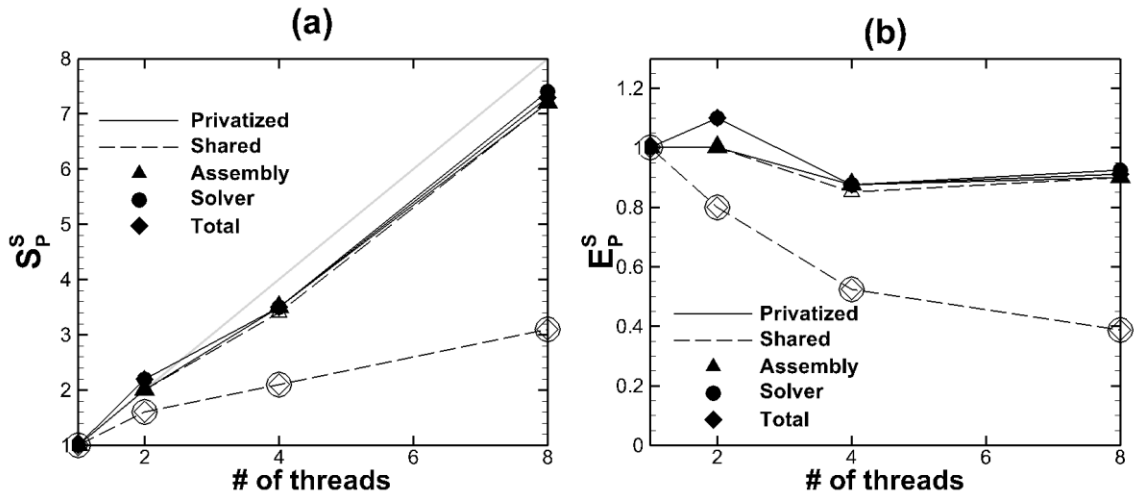

Figure 3.7 Results of speedup and parallel efficiency for steady-state saturated flow case 2 consisting of 10^6 nodes with a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

The parallel efficiency for steady-state flow case 2 with and without privatization is compared in Figure 3.7b. The parallel efficiencies for matrix assembly for both cases are higher than 0.84. When the computing time only for the solver is considered, the parallel efficiency without privatization decreases as the number of threads increases. Parallel efficiency ranges from 0.88 ($= E_4^S$) to 1.10 ($= E_2^S$) for the solver with privatization, and the overall efficiency ranges from 0.88 ($= E_4^S$) to 1.09 ($= E_2^S$). Interestingly, the parallel efficiency is greater than 1.0 when two threads are applied with the privatization method being utilized (super-linear scalable). Taking into account the experimental error, however, ranging from 10 % to 15 %, the result is interpreted to be close to linear scalable computing. Based on the results, the node reordering method applied to this study improves the computational efficiency and the privatization scheme also increases the parallel speedup for steady-state flow case 2 when the code is not optimized for speed.

For steady-state flow case 2 with code-optimization, the overall computing times from serial and parallel computing (T_S and T_P) are listed in Table 3.6. T_S for a serial simulation with natural ordering is 521 sec (Table 3.6). This is about 8 % larger than the mean T_S with node reordering but with a single thread (478 sec). The comparison of T_S values indicates that the difference in T_S for the cases with reordering for the different number of sub-domains is less than 10 % and the T_S values applied to calculate the parallel speedup and efficiency are similar to one another. There is no noticeable relation between T_S and the number of sub-domains.

Table 3.6 Total computing times for serial (T_S) and parallel (T_P) jobs obtained with respect to the number of threads using an optimized version of PHGS for steady-state flow case 2.

Communication Scheme	**T_S (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	520[1)]	484 (336)	501 (209)	476 (176)
Shared		440 (339)	523 (291)	442 (219)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

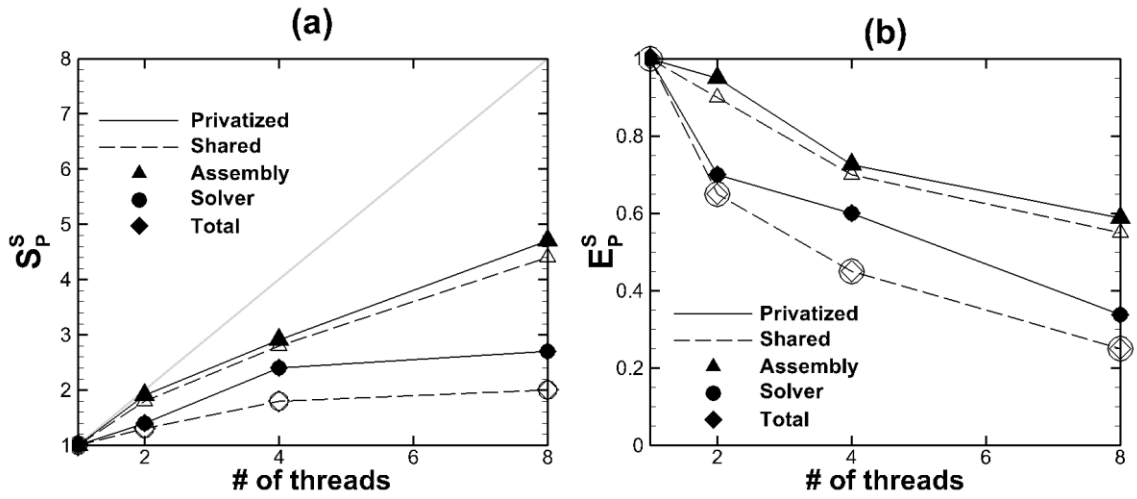

Figure 3.8 Results of speedup and parallel efficiency for steady-state saturated flow case 2 consisting of 10^6 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

For the parallel performance for simulation 2 with code optimization, Figure 3.8 shows the results for parallel speedup (Figure 3.8a) and efficiency (Figure 3.8b) with and without privatization being applied. The maximum parallel speedup with privatization is about 4.7 ($E_8^S = 0.58$) for matrix assembly, 2.7 ($E_8^S = 0.34$) for the solver, and 2.7 ($E_8^S = 0.34$) overall. Similarly, the maximum speedup without privatization is about 4.4 ($E_8^S = 0.56$) for assembly, 2.0 ($E_8^S = 0.25$) for the solver, and 2.0 ($E_8^S = 0.25$) overall. Although the maximum speedup is achieved with 8 threads, the trend in the change of the parallel speedup is slightly different from the case with privatization. Specifically, the major difference is that E_8^S values for the solver and the overall efficiency are about one half of E_2^S and E_4^S.The parallel efficiency for the cases with and without privatization decreases with an increasing number of threads (Figure 3.8b). Similar to the results obtained without code-optimization for speed, the parallel efficiency for global assembly is the highest, but the values are less than that obtained for the cases without optimization. The privatization scheme again improves parallel efficiency.

3.3.3.3 Results for Saturated Flow Simulation 3

Table 3.7 lists the overall computing times obtained from the serial and parallel (T_S and T_P) simulations when Parallel HydroGeoSphere (PHGS) was compiled and optimized for speed. The overall computing time with natural node ordering is about 1.1×10^4 sec which is about two times as much as the mean computing time for the simulations with node reordering and with various numbers of sub-domains (5.8×10^3 sec). The differences in T_S values between the cases with natural ordering and with reordering originate from the number of solver iterations: the natural ordering scheme requires about two times as many iterations as the reordering scheme does.

Table 3.7 Total computing times for serial (T_S) and parallel (T_P) jobs obtained with respect to the number of threads using an optimized version of PHGS for steady-state flow case 3.

Communication Scheme	**T_S (T_P) in sec**				
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**	**16 SDs**
Privatized	11119[1)]	5268 (2422)	5113 (1662)	5512 (778)	5780 (631)
Shared		4821 (3909)	6139 (2255)	7586 (1162)	6059 (1022)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

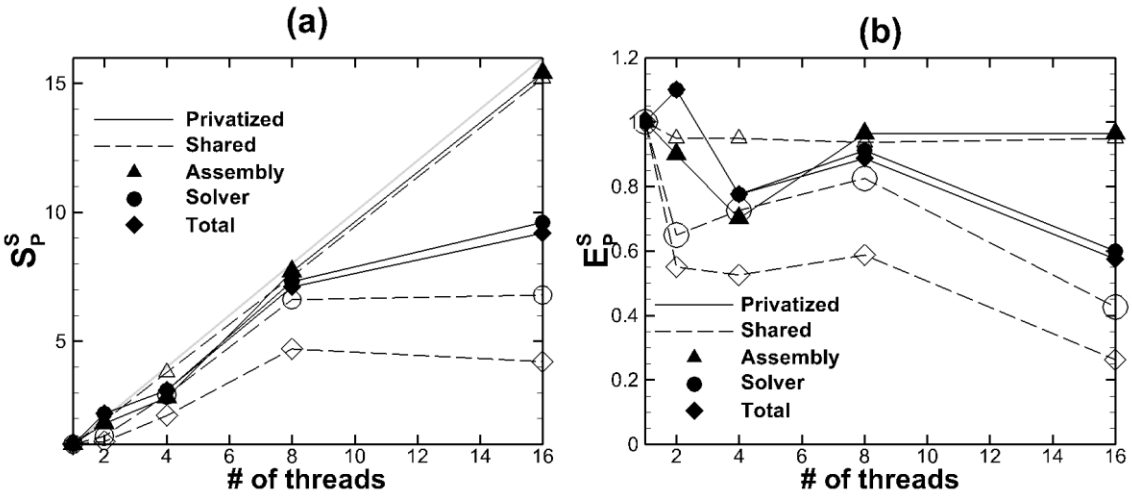

Figure 3.9 Results of speedup and parallel efficiency for steady-state saturated flow case 3 consisting of 10^7 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figures 3.9a and 3.9b illustrate the parallel speedup and efficiency results for simulation 3 when the code is optimized for speed. The maximum speedup with privatization is about 15.4 ($E_{16}^S = 0.96$) for matrix assembly and 9.6 ($E_{16}^S = 0.60$) for the solver. Similarly, the maximum speedup without privatization is about 15.2 ($E_{16}^S = 0.95$) for assembly and 6.8 ($E_8^S = 0.43$) for the solver. The maximum overall speedup is 9.2 ($E_8^S = 0.57$) with privatization and it is about 6.6 ($E_8^S = 0.41$) without privatization. The mean parallel efficiency for matrix assembly is 0.91 and 0.96 with and without privatization, respectively. The parallel speedup is improved with up to 16 threads with privatization while it is not improved without privatization. The saturated flow simulation is a solver intensive type, for which the matrix solver takes more than 95 % of total computing time, and thus the parallel efficiency of the matrix solver controls the total parallel efficiency. Based on the evaluation tests for the saturated flow simulations, the privatization scheme shows a higher parallel efficiency for the matrix solve than does the shared scheme. Specifically, if the privatization scheme is used, the parallel efficiency of the matrix solver is about 50 % more than that obtained with the shared scheme. The privatization scheme consistently improves the computational efficiency for all numbers of threads applied in this study.

Table 3.8 Simulation cases for weak scaling tests under steady-state saturated flow

Thread	Number of nodes (nx × ny × nz)	
	GPC[1)]	TCS[2)]
1	100×100×10	100×1000×10
2	100×200×10	100×2000×10
4	100×400×10	100×4000×10
8	100×800×10	100×8000×10
16	NA	100×16000×10

GPC[1)] : General Purpose Cluster at SciNet/University of Toronto
TCS[1)] : Tightly Coupled System at SciNet/University of Toronto

3.3.4 Weak Scaling Tests for Steady-State Saturated Flow

Under the same simulation conditions as in the strong scaling tests, weak scaling tests were performed using two different scaling compartments (see Figure 3.2). A smaller compartment having 10^5 nodes is used on GPC with up to 8 threads, and thus the number of nodes ranges from 1×10^5 to 8×10^5. A larger compartment with the 10^6 number of nodes is used on TCS using up to 16 threads, and thus the number of equations solved for the problem ranges from 1.0×10^6 to 1.6×10^7 (Table 3.8). For all the problems tested for weak scaling, PHGS was compiled for optimal speed.

Figures 3.10a and 3.10b show the results of the weak scaling tests for steady-state saturated flow simulations performed on GPC and TCS, respectively. The parallel efficiency of weak scaling, E_p^W, for matrix assembly, the matrix solver and overall were calculated using Equation (4). To compute an overall E_p^W, only the time taken for the computational tasks consisting of matrix assembly, preconditioning and matrix solution is considered. E_P^W for assembly, the solver and overall on GPC gradually decreases with an increasing number of threads. Specifically, E_2^W is about 0.83 for assembly and 0.74 for the solver, while E_8^W drops to 0.26 for assembly and 0.19 for the solver.

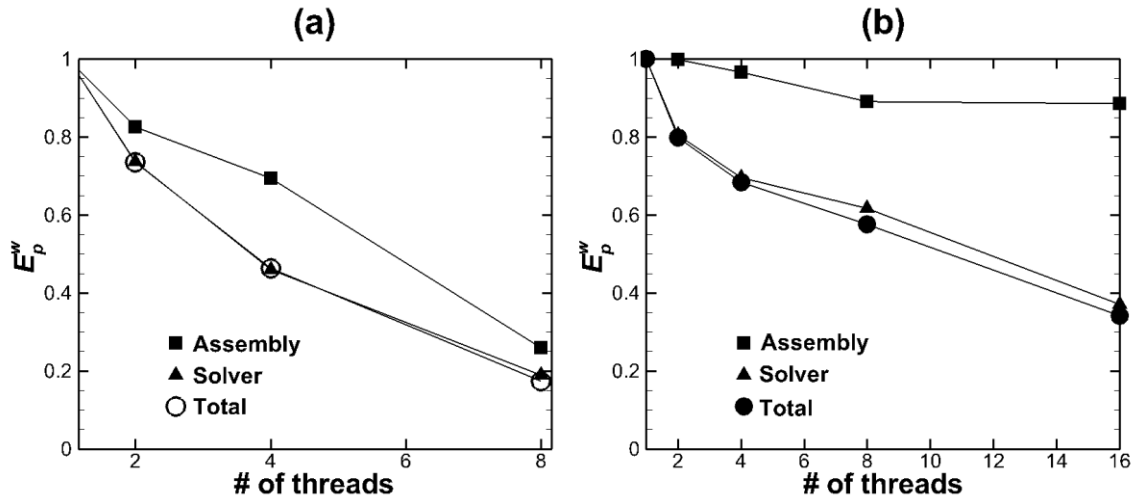

Figure 3.10 Results of weak scaling tests for steady-state saturated flow performed on a) GPC and b) TCS

The behaviour of E_P^W when performed on TCS is similar to that noted for GPC, in that E_P^W is inversely proportional to the number of threads. However, E_P^W on TCS for matrix assembly is higher, 0.89 (=E_{16}^W). The decreasing ratio of E_P^W to the number of threads is lower than that obtained on GPC. The minimum parallel efficiency, E_{16}^W, is 0.37 for the solver and 0.34 for the overall computation. Thus, TCS shows slightly higher parallel efficiency for saturated steady-state flow, which is a solver intensive problem.

One of the reasons for the low parallel efficiency is because the computing time for serial computing increases exponentially as the size of the problem becomes larger. For example, if the problem size increases by a factor of eight, the computing time with a single thread increases by more than 12 times. If the two parallelized parts are compared, matrix assembly shows a relatively high parallel efficiency mainly because data dependency is lower for the assembly step.

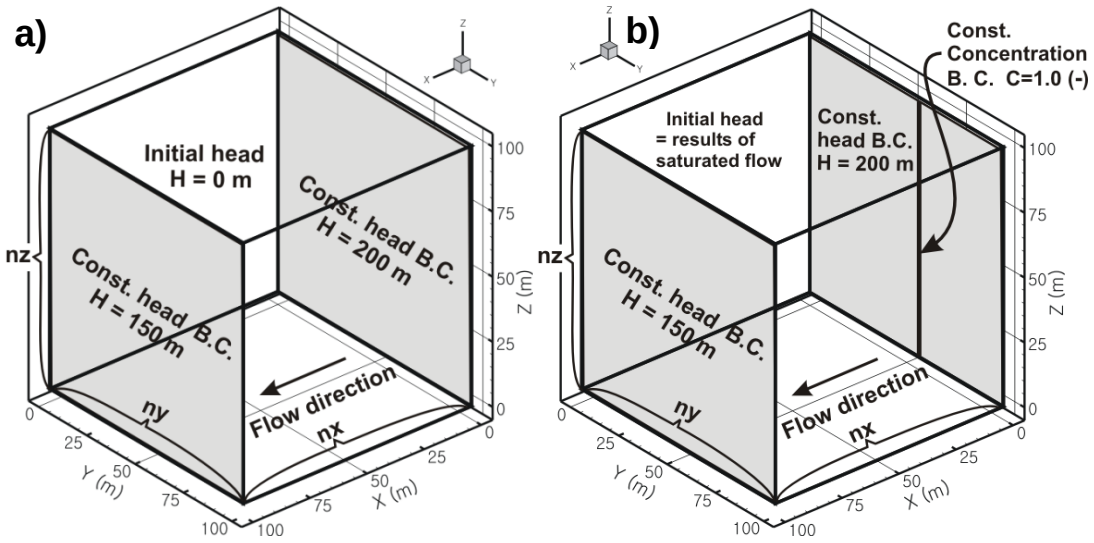

Figure 3.11 Schematic of simulation domain for a) steady-state saturated flow and b) saturated flow and contaminant transport

3.4 Parallel Efficiency for Saturated Flow and Transport

3.4.1 Simulation Description and Domain Partitioning

To evaluate the parallel efficiency for solute transport, steady-state saturated groundwater flow and transient contaminant transport are simulated in a three-dimensional heterogeneous domain of size 100 m × 100 m × 100 m (Figure 3.11). For the flow problem, specified heads equal to H = 200 m and H = 150 m are assigned on the right (x= 0 m) and left (x= 100 m) sides of the domain, respectively, with the remaining boundaries taken to be impermeable (Figure 3.11a). For transport, a specified concentration ($C = 1.0$) is assigned to a vertical line of nodes located along the z-axis at $x = 0$ m and $y = 50$ m.

Table 3.9 Simulation cases for evaluating parallel efficiency for transient contaminant transport

No	Number of nodes (nx × ny × nz)	Heterogeneity		Test machine
		Mean ln(K[1)])	Var ln(K)	
1	100 × 100 × 10	-4.7	1.5	GPC[2)]
2	330 × 330 × 10	-4.9	2.0	GPC
3	3300 × 330 × 10	-4.6	0.0	TCS[3)]

K[1)]: Hydraulic conductivity (m/day).
GPC[2)]: General Purpose Cluster at SciNet/University of Toronto
TCS[3)]: Tightly Coupled System at SciNet/University of Toronto

Parallel efficiency is evaluated with the same grid resolutions as was used for the saturated flow simulations provided in the previous section. Table 3.9 summarizes the computational grid and the heterogeneity of the hydraulic conductivity field used in the simulations. The ln K variance is 1.5 for simulation 1 and 2.0 for simulation 2. The horizontal and vertical correlation lengths are 50 m and 10 m for simulation 1 and 10 m and 1 m for simulation 2. Simulation 3 was performed using a homogeneous conductivity field with the value of $K = 10^{-2}$ m/day. For all simulations, the values of the simulation parameters are a porosity equal to 0.1, longitudinal and transverse dispersivities equal to 10^{-1} m and 10^{-3} m, respectively, and an effective diffusion coefficient equal to 10^{-10} m^2/sec. Simulations 1 and 2 were performed on GPC, and simulation 3, a relatively large problem, was performed on TCS. The reported data describing parallel performance considers only the transport part of the problem.

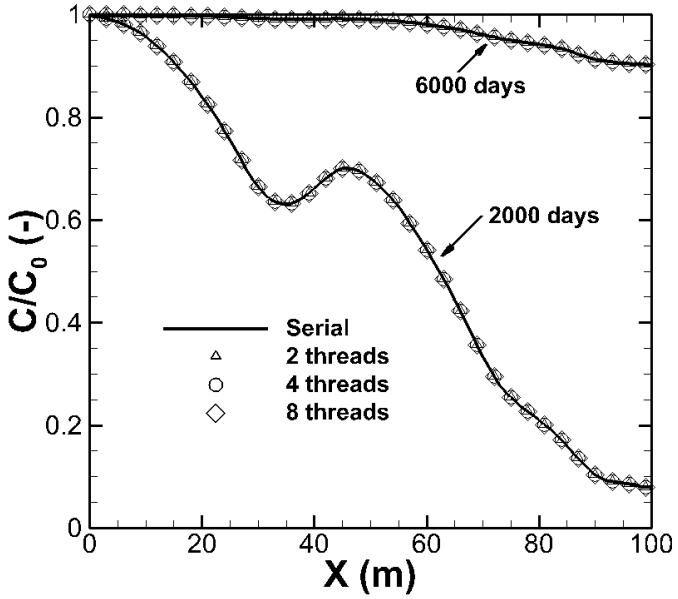

Figure 3.12 Comparison of concentration profiles for simulation 1 between serial and parallel computations with elapsed time of 2000 and 6000 days. Profile is located at y = 50 m and z = 50 m.

3.4.2 Consistency of Parallel Simulations

It is necessary to investigate the accuracy and consistency of the transport solutions by parallel computing before evaluating the parallel computational efficiency. Parallel simulations were performed on GPC with 2, 4 and 8 threads and on TCS with 2, 4, 8 and 16 threads. Figure 3.12 shows the concentration profiles for simulation 1 at simulation times of 2000 and 6000 days along a profile located along the *x*-axis at *y* = 50 m and *z* = 50 m. Similar to the saturated flow simulations, the concentration distribution is irregular due to the heterogeneous hydraulic conductivity field. The maximum difference in concentration between serial and parallel simulations is less than 10^{-5} %, and thus it is concluded that the simulation results from parallel computing are consistent with the serial results.

Table 3.10 Comparison of the number of solver iterations for transient solute transport cases.

Thread	Simulation 1 (29484)[1)]		Simulation 2 (19162)[1)]		Simulation 3 (20737)[1)]	
	Privatized	**Shared**	**Privatized**	**Shared**	**Privatized**	**Shared**
1	44637	47429	22472	22810	33553	33707
2	41021	45967	21992	22540	32677	32507
4	47416	49999	22872	23020	32738	33033
8	48112	50420	22922	23107	34090	33196
16	NA	NA	NA	NA	35884	35584

[1)]: Number of iterations in serial computing without reordering

Table 3.10 lists the total number of solver iterations taken to perform transport simulations 1, 2 and 3. The number of iterations in the brackets in Table 3.10 for each simulation was obtained from serial computing without domain partitioning and with natural ordering. For the case of simulation 1, it was found that the number of solver iterations varied with the number of threads used even though the transport solutions from the serial and parallel simulations are the same. Specifically, the number of iterations for the serial simulation is about 2.9×10^4, while that for parallel simulations with privatization ranges from 4.1×10^4 to 5.0×10^4. Parallel simulations take about 59 % more iterations than the serial case. For simulation 2, the difference in the number of solver iterations between the serial and parallel simulations is relatively minor. The number of iterations for the serial simulation is 1.9×10^4, and that for the parallel simulations with node reordering is about 2.3×10^4. Thus, the parallel computation requires about 19 % more solver iterations than that for serial computing. The results for simulation 3, which was performed on TCS, are similar to those obtained for simulations 1 and 2. The mean number of iterations taken for the parallel simulation both with and without privatization is 3.4×10^4. It is noted that the reordering scheme used in this study for partitioning the simulation domains does not directly affect the number of iterations.

3.4.3 Results of Strong Scaling Tests for Solute Transport

Equations (3.1) and (3.2) are applied to evaluate the parallel efficiency and speedup for the transient solute transport simulations. Parallel efficiency is compared with and without privatization being applied and when the code is or is not optimized for speed.

Figures 3.13 to 3.17 show the results for parallel speedup, S_P^S, for the cases with and without privatization and performed on GPC and TCS: the solid line with the closed symbols is with privatization and the dotted line with the open symbols is without privatization. The parallel speedups for the various different computing steps such as matrix assembly (triangles) and solver (circles) are evaluated separately and compared to each other.

3.4.3.1 Results for Solute Transport Simulation 1

Table 3.11 lists the values of T_s obtained from serial computing with natural ordering and also when the nodes are reordered with various numbers of sub-domains, when Parallel HydroGeoSphere (PHGS) is not optimized for speed. The computing time with natural ordering is 3375 sec, or about 15 % smaller than the mean computing time with node reordering (3871 sec). The efficiency with privatization for the serial simulations is consistently higher than the cases without privatization but the difference among them is less than 6 %. The T_s values for the both cases with and without privatization increase with the increasing number of threads.

Table 3.11 Total computing times for serial (T_S) and parallel (T_P) jobs obtained with different numbers of threads for transient simulation 1 using a non-optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	3375[1)]	3645 (1934)	3827 (1152)	3868 (634)
Shared		3783 (2264)	3990 (1524)	4110 (1014)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

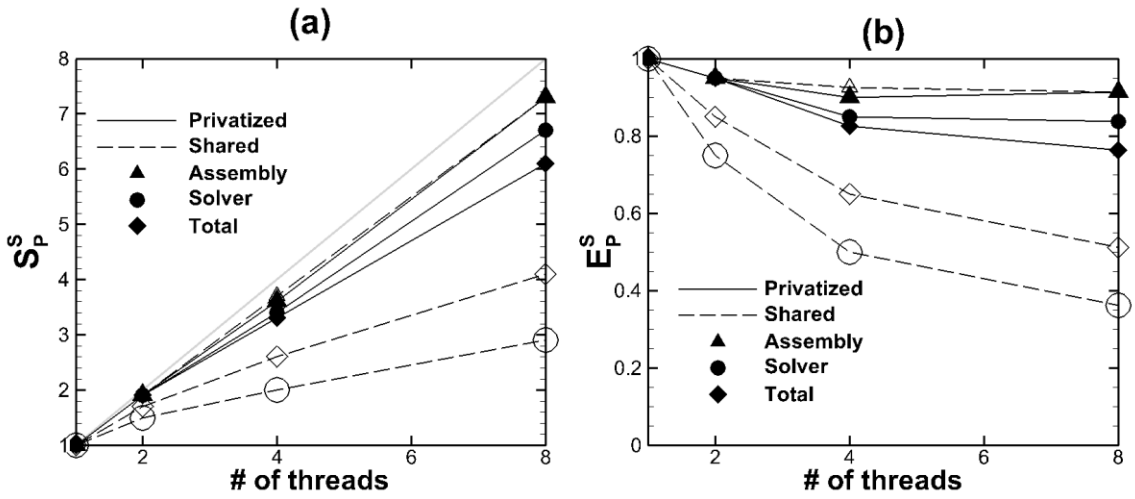

Figure 3.13 Results of speedup and parallel efficiency for transport simulation 1 consisting of 10^5 nodes with a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.13 shows the results for (a) parallel speedup, S_P^S, and (b) parallel efficiency, E_P^S, on GPC, repectively. The parallel speedup for matrix assembly (triangles) and the matrix solver (circles) are also compared with each other to evaluate which component is more efficient when the privatization scheme is applied. The maximum speedup with privatization is about 7.3 ($E_8^S = 0.91$) for matrix assembly and it is 6.7 ($E_8^S = 0.84$) for the matrix solver. Similarly, the maximum speedup without applying privatization is about 7.3 ($E_8^S = 0.91$) for assembly and 2.9 ($E_8^S = 0.37$) for the solver. The maximum overall speedup is 6.1 ($E_8^S = 0.76$) with privatization and it is about 4.1 ($E_8^S = 0.51$) without privatization. Although the efficiency deceases with the number of threads for the cases with and without privatization, E_P^S with privatization is greater than 0.76 while E_P^S without privatization can be as low as 0.51.

Table 3.12 Total computing times for serial (T_S) and parallel (T_P) jobs obtained with different numbers of threads for transport simulation 1 using an optimized version of PHGS

Communication Scheme	T_s (T_P) in sec			
	1 SD	2 SDs	4 SDs	8 SDs
Privatized	1227[1)]	1385 (790)	1385 (557)	1555 (403)
Shared		1393 (804)	1527 (565)	1506 (502)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

For the cases where PHGS is optimized for speed when it is compiled (Table 3.12), the strong scaling results for simulation 1 differ from those obtained without code optimization (Table 3.11). The overall computing times from the serial simulations (T_S) are listed in Table 3.12 when the model is compiled and optimized for speed. T_S values obtained with the different numbers of sub-domains range from 1227 to 1554 sec, and the mean T_S with reordering is 1458 sec. T_S with natural ordering is 1227 sec, which is about 84 % of the mean T_S obtained with reordering. When PHGS is optimized for speed, serial simulations with natural ordering are relatively fast. When reordering is applied, the simulations with privatization are faster than those without privatization as was the case for the saturated flow simulations described in the previous section.

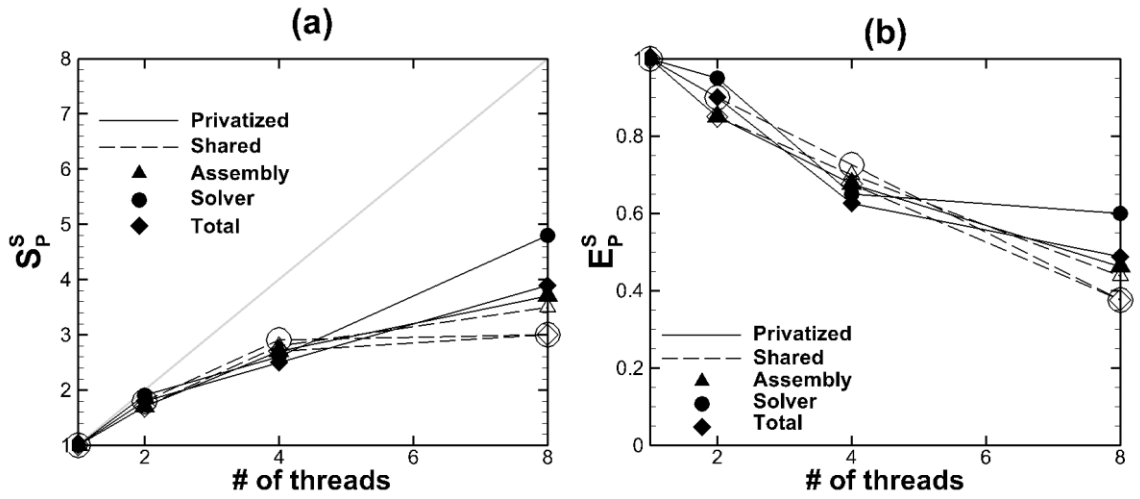

Figure 3.14 Results of speedup and parallel efficiency for transport simulation 1 consisting of 10^5 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.14 shows the results for parallel speedup and efficiency obtained with the optimized version of PHGS. The speedup with an increasing number of threads is similar to that obtained in the previous examples. The maximum S_P^S is obtained when 8 threads are applied. The maximum speedup S_8^S without privatization is 3.5 ($E_8^S = 0.44$) for matrix assembly and it is 3.0 ($E_8^S = 0.38$) for the solver, and the maximum overall speedup is 3.0 ($E_8^S = 0.37$). When privatization is applied, S_8^S is 3.7 ($E_8^S = 0.46$) for assembly, 4.8 ($E_8^S = 0.61$) for the solver, and 3.9 ($E_8^S = 0.48$) for the overall computation. Although the efficiency for the cases with and without privatization declines with an increasing number of threads used, it is consistently higher with privatization compared to the case without privatization.

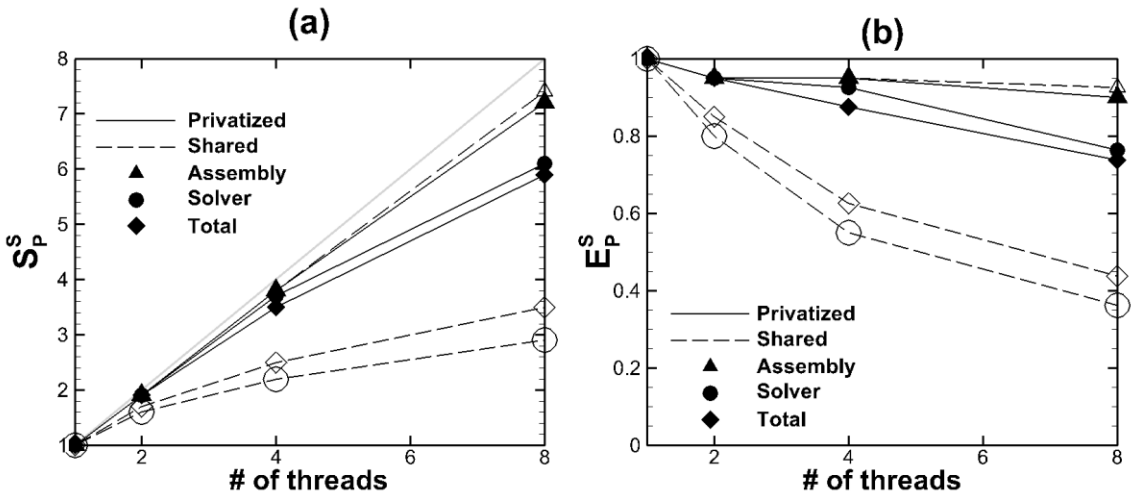

Figure 3.15 Results of speedup and parallel efficiency for transport simulation 2 consisting of 10^6 nodes with a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Table 3.13 Total computing times for serial (T_s) and parallel (T_P) jobs, obtained with different numbers of threads for transport simulation 2 using a non-optimized version of PHGS

Communication Scheme	T_s (T_P) in sec			
	1 SD	2 SDs	4 SDs	8 SDs
Privatized	11399[1)]	11479 (6185)	11736 (3313)	12063 (2042)
Shared		12607 (7540)	13094 (5260)	13008 (3673)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

3.4.3.2 Results for Solute Transport Simulation 2

With Parallel HydroGeoSphere (PHGS) compiled without optimization for speed, Table 3.13 lists T_s values obtained from serial computing with the natural ordering and reordering schemes. The computing time with natural ordering is similar to that obtained with node reordering and up to 8 sub-domains. For the cases with reordering, T_s with privatization is less than that without privatization. However, the difference in T_s values with and without privatization is less than 10 %.

Figure 3.15 compares the parallel speedup and parallel efficiency with and without privatization. The parallel speedup for both cases increases with the number of threads used. The maximum speedup with the privatization approach is about 7.2 ($E_8^S = 0.90$) for matrix assembly and

it is 6.1 ($E_8^S = 0.76$) for the solver. Similarly, the speedup without privatization is about 7.4 ($E_8^S = 0.93$) for assembly and 2.9 ($E_8^S = 0.37$ for the solver. Overall speedup with the privatization scheme is about 5.9 ($E_8^S = 0.74$), and it is about 3.5 ($E_8^S = 0.44$) without privatization. From the results shown in Figure 3.15, the parallel efficiency deceases with the number of threads for the cases with and without privatization. If 8 threads are applied, the matrix solver with the privatization approach is about 2.1 times more efficient compared to the solver without privatization.

Table 3.14 Total computing times for serial (T_s) and parallel (T_P) jobs obtained with different numbers of threads for transport simulation 2 using an optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	3826[1)]	3990 (2381)	4228 (1528)	4106 (1377)
Shared		4124 (2592)	4275 (1776)	4110 (1526)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

For the results of simulation 2 with PHGS being optimized for speed, the overall serial computing time (T_s) obtained with the reordering scheme is slightly larger than that from serial computing with natural ordering (). With node reordering, T_s does not necessarily increase as the number of partitioned domains increases. Compared to the cases without privatization, the influence of the privatization scheme on the computing time is insignificant for these simulations.

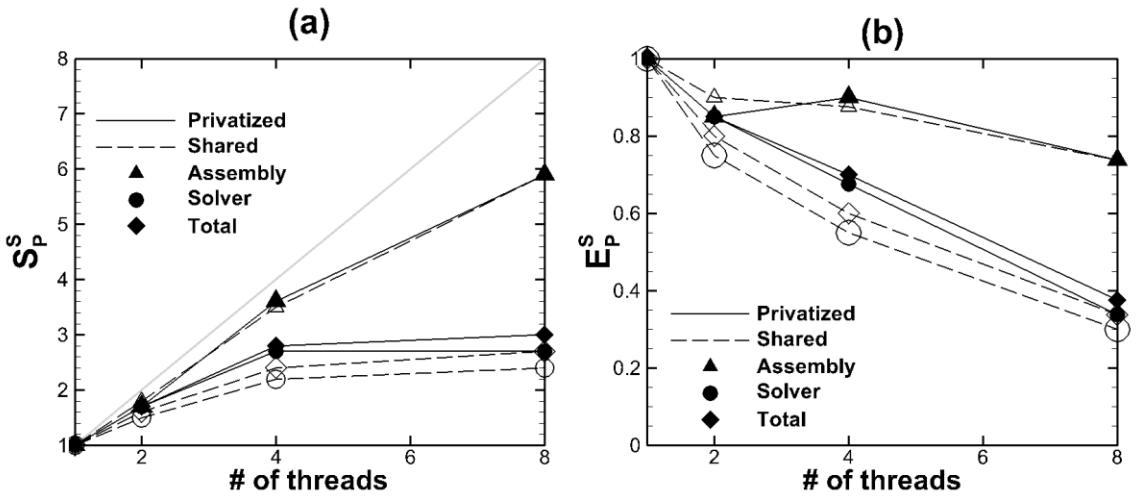

Figure 3.16 Results of speedup and parallel efficiency for transport simulation 2 consisting of 10^6 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.16 illustrates the results for the parallel speedup and the parallel efficiency for simulation cases with and without privatization when an optimized PHGS for speed is used. The speedup with the number of threads used shows a similar behavior to those in the previous examples: S_P^S monotonically increases with the number of threads and the maximum S_P^S is obtained when 8 threads are applied. Specifically, the maximum S_P^S value with privatization is about 5.9 ($E_8^S = 0.74$) for matrix assembly and 2.7 ($E_8^S = 0.34$) for the solver, and it is about 5.9 ($E_8^S = 0.74$) for matrix assembly and 2.4 ($E_8^S = 0.30$) for the solver with the shared scheme. The maximum overall speedup is about 3.0 ($E_8^S = 0.37$) with privatization and about 2.7 ($E_8^S = 0.34$) without privatization.

The parallel efficiency deceases with the number of threads for the cases with and without privatization. The efficiency is higher for matrix assembly compared to that for the solver. Similar to the results for the previous simulations, the privatization scheme improves the efficiency.

3.4.3.3 Results for Solute Transport Simulation 3

Table 3.15 lists the overall computing times, T_s, from serial computing when Parallel HydroGeoSphere (PHGS) is optimized for speed on TCS. The T_s values obtained with natural ordering and with the reordering scheme for up to 16 sub-domains are about 9.9×10^4 sec and 1.4×10^5 sec, respectively. For the serial computations involving simulation 3, the natural ordering scheme is about 1.2 times more efficient than the reordering scheme. This result indicates that simulations with reordering can be less efficient than those with natural ordering, which is contrary to the previous simulation results. The difference in T_s with and without privatization is more than 10 %. Based on the T_s differences, the privatization scheme improves computing efficiency in parallel computing.

Table 3.15 Total computing times for serial (T_s) and parallel (T_P) jobs obtained with different numbers of threads for transport simulation 3 using an optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**				
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**	**16 SDs**
Privatized	98876[1)]	109421 (71649)	109134 (40208)	116623 (24563)	116969 (18084)
Shared		123805 (87059)	122602 (51039)	124769 (34476)	135359 (29748)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

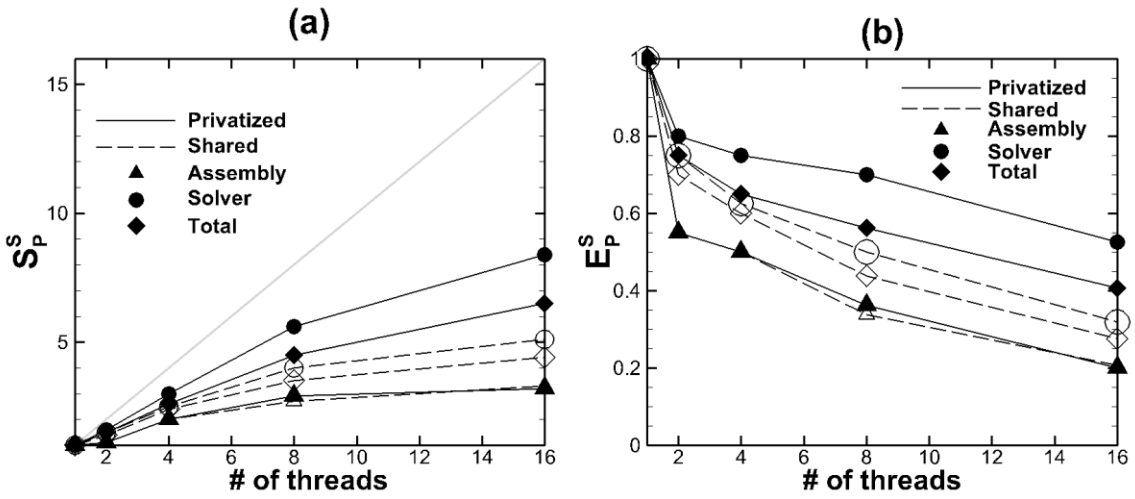

Figure 3.17 Results of speedup and parallel efficiency for transport simulation 3 consisting of 10^7 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.17 illustrates the parallel speedup and efficiency for simulation 3. The maximum speedup with privatization is about 3.2 ($E_{16}^{S} = 0.20$) for matrix assembly and 8.4 ($E_{16}^{S} = 0.53$) for the matrix solver. Similarly, the maximum speedup is about 3.3 ($E_{16}^{S} = 0.20$) for assembly and 5.1 ($E_{16}^{S} = 0.32$) for the solver without privatization. The maximum overall speedup with the privatization scheme is about 6.5 ($E_{16}^{S} = 0.41$) and it is about 4.4 ($E_{16}^{S} = 0.28$) without privatization. The mean parallel efficiency for matrix assembly is 0.52 with the privatization scheme and 0.60 without privatization. Interestingly, the parallel speedup for matrix assembly ranged from 1.1 ($E_{2}^{S} = 0.55$) to 3.3 ($E_{16}^{S} = 0.20$) for the cases with and without privatization. This behavior is different compared to those obtained for simulations 1 and 2 in which the parallel efficiency for matrix assembly is around 0.89 on average. The relatively low parallel efficiency for matrix assembly reduces the overall parallel speedup. However, the parallel efficiency for the matrix solver and the overall efficiency are improved by applying the privatization approach.

3.4.4 Weak Scaling Tests for Solute Transport

To evaluate parallel efficiency with weak scaling tests, a three-dimensional homogeneous domain of size 100 m × 100 m × 100 m with 100 × 100 × 11 nodes is used as a scaling compartment for GPC. The simulation conditions applied to the weak scaling tests are the same as those as in the strong scaling tests. For the weak scaling tests involving for transient solute transport, simulations were performed using grid sizes ranging from 1×10^5 to 8×10^5 nodes and were performed on GPC. The simulations performed with larger numbers of nodes, ranging from 1×10^6 to 1.6×10^7, were performed on TCS (Table 3.16). The number of nodes increases proportionally to the number of threads used in the weak scaling tests. Specifically, the number of grid lines along the *y*-axis was increased with the increasing number of threads. A homogeneous hydraulic conductivity field ($K = 10^{-2}$ m/day) was used for all the simulations and the same boundary conditions were applied as in the simulations used for the strong scaling tests.

Table 3.16 Transient transport simulation cases for evaluating weak parallel efficiency

Number of thread	Number of nodes (nx × ny × nz)	
	GPC[1)]	TCS[2)]
1	100 × 100 × 11	100 × 1000 × 11
2	100 × 200 × 11	100 × 2000 × 11
4	100 × 400 × 11	100 × 4000 × 11
8	100 × 800 × 11	100 × 8000× 11
16	NA	100 × 16000× 11

GPC[1)]: General Purpose Cluster at SciNet/University of Toronto
TCS[2)]: Tightly Coupled System at SciNet/University of Toronto

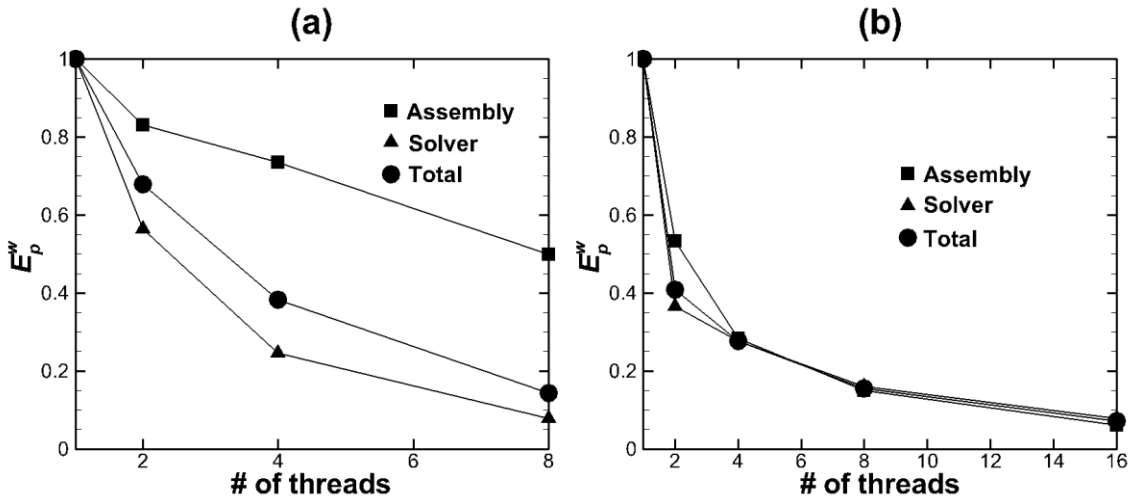

Figure 3.18 Results of weak scaling tests for solute transport performed by a) GPC and b) TCS

Figure 3.18 summarizes the results of the weak scaling tests performed on GPC (Figure 3.18a) and TCS (Figure 3.18b). The parallel efficiency for the weak scaling tests, E_p^w, decreases as the number of threads increases. Specifically, for the simulations performed on GPC, E_p^w for matrix assembly is 0.83 with 2 threads, 0.74 with 4 threads, and 0.50 with 8 threads. The decreasing trend in E_p^w values for the matrix solver is stronger than that for assembly: E_2^w, E_4^w, and E_8^w for the solver is 0.56, 0.25, and 0.08, respectively.

For the parallel efficiency on TCS (Figure 3.18b), E_p^w decreases drastically to 0.53 for matrix assembly and to 0.36 for the solver when 2 threads are used. E_{16}^w is less than 0.1 for both matrix assembly and the solver. Thus, the parallel efficiency for this type of simulations is relatively low compared to the efficiency obtained for transport simulations 1 and 2.

3.5 Parallel Efficiency for Transient Variably-Saturated Flow Simulations

3.5.1 Simulation Description and Domain Partitioning

Similar to the steady-state saturated flow simulations, transient variably-saturated groundwater flow was simulated in a three-dimensional domain of size 100 m × 100 m × 100 m and with various numbers of nodes. The grid sizes used to evaluate parallel efficiency are also the same as those used for the saturated flow cases. Simulations 1 and 2 involved 10^5 and 10^6 nodes and heterogeneous hydraulic conductivity fields, while the mesh for simulation case 3 consisted of 10^7 nodes and a homogeneous conductivity field. Simulations 1 and 2 were performed on GPC and simulation 3 was performed on TCS (Table 3.17). The simulation domain consists of two flow regimes: fully-saturated and vadose zones. Above the fully-saturated region, a partially-saturated vadose zone extends to the top of the domain (Figure 3.19). A specified head of 50 m is applied to the four sides of the domain and a specified flux condition is applied at the top boundary ($Q = 8.1 \times 10^{-3}$ m/day). The bottom boundary is taken to be impermeable. An observation line located at y = 50 m and z = 0 m is set at the bottom of the domain to compare the solutions obtained from serial and parallel computing.

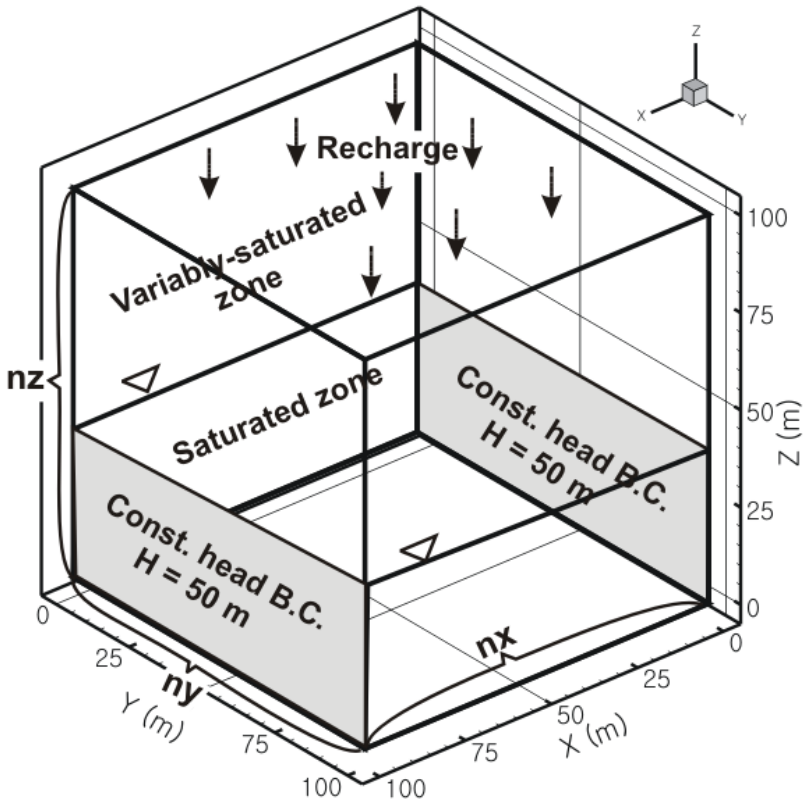

Figure 3.19 Schematic of the simulation domain for variably-saturated flow

Table 3.17 Simulation cases for evaluating parallel efficiency for variably-saturated flow

no	Number of node (nx × ny × nz)	Heterogeneity		Test machine
		Mean ln(K)[1)]	Var ln(K)	
1	100 × 100 × 10	-4.6	9.9	GPC[2)]
2	330 × 330 × 10	-4.6	9.9	GPC
3	3300 × 330 × 10	-4.6	0.0	TCS[3)]

K[1)]: Hydraulic conductivity (m/day).

GPC[2)] : General Purpose Cluster at SciNet/University of Toronto

TCS[3)] : Tightly Coupled System at SciNet/University of Toronto

For simulations 1 and 2, both the fully- and partially-saturated zones are heterogeneous and the hydraulic conductivity fields follow a lognormal distribution (with a geometric mean and a variance of ln (K) euqual to -4.6 and 9.9, respectively) and with exponential correlations (horizontal correlation length equals to 10.0 m and the vertical value is 2.0 m). The van Genuchten model is used for the soil moisture characteristic curve and relative permeability-saturation-head relation. The parameters relevant to the unsaturated zone are listed inTable 3.18.

Table 3.18 Simulation parameters for the unsaturated zone

Porous medium property	**Value**
Porosity [-]	0.1
Unsaturated Van Genuchten functions	
α [1/m]	3.34
β [-]	1.982
Residual saturation [-]	0.2771

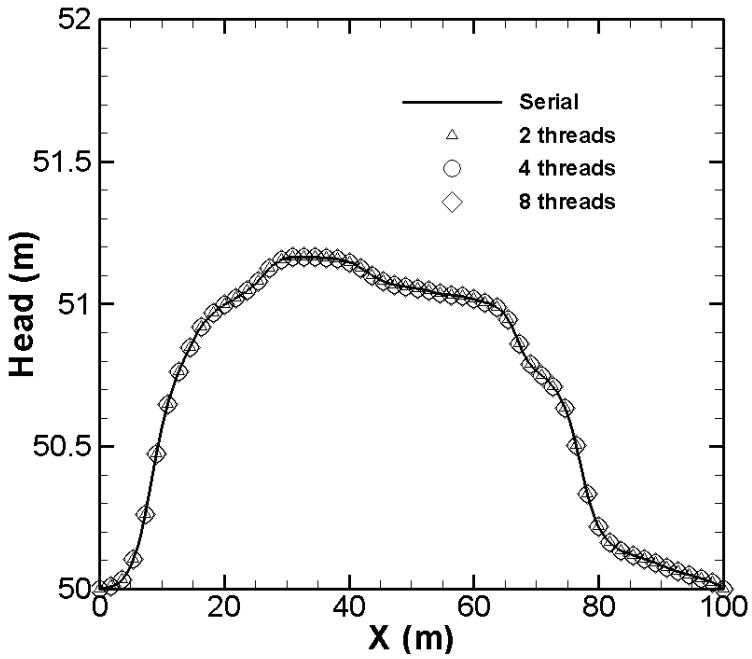

Figure 3.20 Simulation results comparing heads at steady state along a profile obtained with serial and parallel computations for the variably-saturated flow problem. The profile is located at a y = 50 m and z = 0 m.

3.5.2 Consistency of Parallel Simulations

The accuracy and consistency of the solutions obtained with parallel computing are investigated first before evaluating the parallel efficiency for the variably-saturated flow simulations. The distributions of hydraulic heads along the observation line are compared from the serial and parallel computing results for various numbers of threads (Figure 3.20). The hydraulic head distribution indicates that the solutions are consistent and that the parallel implementation is valid. Watertable mounding caused by the infiltration assigned at the top of the domain is observed and the heads at the right and left sides of the simulation domain are the same as the specified value, 50 m. The irregular shape of the groundwater mounding was due to the heterogeneous hydraulic conductivity distribution. From the comparisons provided in Figure 3.20, the maximum difference in the head solution between the serial and parallel simulations is less than 10^{-3} m, which is the same order of magnitude assigned for the tolerance of the Newton-Raphson convergence.

Table 3.19 Comparison of the number of iterations for variably-saturated flow cases.

Thread	Simulation 1 (17494)[1)]		Simulation 2 (20953)[1)]		Simulation 3 (15389)[1)]	
	Privatized	**Shared**	**Privatized**	**Shared**	**Privatized**	**Shared**
1	15769	15781	18348	18403	13202	12709
2	15367	15543	18227	18234	12531	10725
4	15687	16011	18217	18414	12652	11688
8	16358	16349	18623	18661	15477	15090
16	NA	NA	NA	NA	13151	11979

[1)]: Number of iterations in serial computing without reordering

The number of solver iteration taken for each simulation is listed in Table 3.19. Similar to the results for saturated flow, the node reordering scheme reduces the number of solver iteration and the number of solver iterations with natural ordering is consistently more than that with the reordering scheme. Specifically, simulations 1 and 2 required a mean number of iterations equal to 15858 and 18391 using the reordering scheme, while with natural ordering it was about 17494 and 20953 on average, respectively. For simulation 3, in which a homogeneous conductivity field is used, the mean number of solver iteration was 12920 using the reordering method, and 15389 with natural ordering to reach a pseudo steady-state flow condition. Thus, the reordering scheme applied is consistently more cost effective than the natural ordering scheme. Based on the results of the comparison, the reordering method can reduce the number of iterations by about 12 %.

With an increasing number of threads used for parallel computing, the number of solver iterations increases. The increase in the number of iterations is partially because of the increasing number of boundary nodes between the partitioned domains, which requires additional communication between the threads. For the case with 16 partitioned domains, the number of solver iterations declined by about 18 % compared to the serial simulation (Table 3.19).

3.5.3 Results of Strong Scaling Tests

3.5.3.1 Results for Transient Variably-Saturated Flow Simulation 1

Table 3.20 lists the serial (T_s) and parallel (T_P) computing times for the natural ordering and reordering schemes when Parallel HydroGeoSphere (PHGS) is compiled without the optimization option. The T_s value obtained using the natural ordering scheme is 1780 sec, and the mean T_s from the simulations with reordering is 1540 sec. The difference between the two cases is about 13 %. Based on the results, the reordering scheme reduces the total computing time. There is a trend in that the T_s value increases as the number of sub-domains increases. Although the T_s values for the simulations with privatization are less than those without privatization, the difference in T_s values with and without privatization ranges from 1 % to 2 %, which is insignificant.

Table 3.20 Total computing times T_s and T_P for variably-saturated flow with different numbers of threads using a non-optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	1780[1)]	1506 (808)	1519 (441)	1563 (286)
Shared		1522 (962)	1539 (558)	1590 (403)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

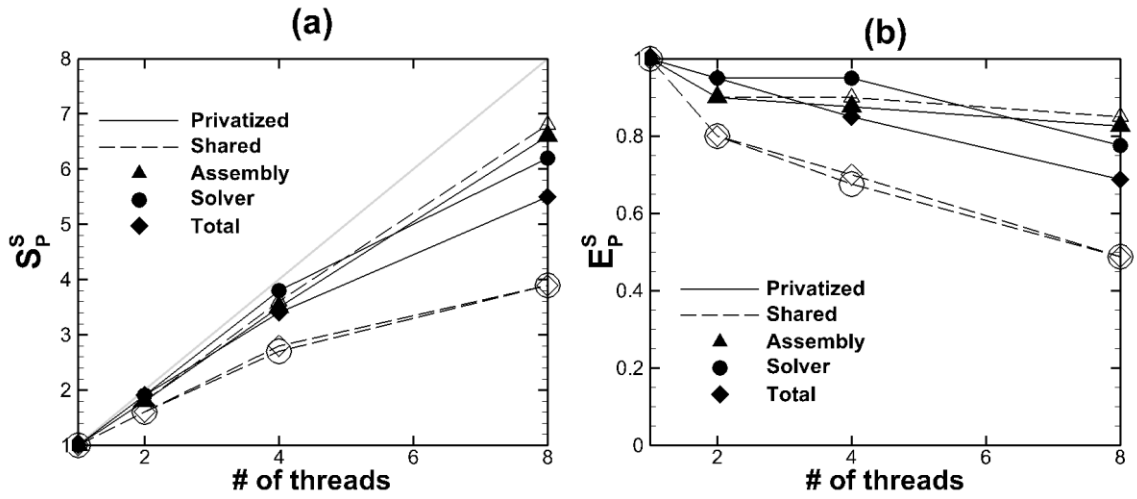

Figure 3.21 Results of speedup and parallel efficiency for variably-saturated flow simulation 1 consisting of 10^5 nodes and a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.21 shows the parallel speedup and efficiency results for simulations 1 without code optimization for speed. The parallel speedup for simulations both with and without privatization increases with the number of threads used. The maximum speedup with privatization is about 6.6 ($E_8^S = 0.82$) for matrix assembly and it is 6.2 ($E_8^S = 0.78$) for the solver when eight threads are used. Without privatization, the maximum speedup is about 6.8 ($E_8^S = 0.85$) for assembly and 3.9 ($E_8^S = 0.49$) for the solver. Maximum overall speedup with privatization is 5.5 ($E_8^S = 0.68$), and about 3.9 ($E_8^S = 0.49$) without privatization.

The parallel efficiency for the cases with and without privatization decreases with an increasing number of threads, although the rate of efficiency reduction is much lower with privatization. Specifically, with the privatization scheme being applied, E_2^S and E_4^S for the matrix solver are 0.97 and 0.95, respectively. These values are close to linear scalability. Without privatization, E_2^S and E_4^S for the solver are 0.79 and 0.69, respectively. Thus, the efficiency of parallel computing can again be improved by applying the privatization scheme.

Table 3.21 Total computing times for serial (T_S) and parallel (T_P) jobs for variably-saturated flow simulation 1 with different numbers of threads using an optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	598[1)]	551 (345)	569 (208)	590 (175)
Shared		543 (376)	569 (285)	588 (266)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

As noted from the previous test results, when PHGS is optimized for speed, the simulation time is about one third compared to the cases without optimization. Table 3.21 lists the total computing times with a single thread, T_S, and multiple threads, T_P, for the natural ordering and reordering schemes. The T_S values obtained using the natural ordering scheme is 598 sec, while the T_S values with the reordering scheme are on average 569 sec. Although the reordering scheme improves the computing efficiency even in serial simulations, the difference between the two cases is less than 5 %. For the cases with reordering, there was a tendency that T_S increases with the number of sub-domains, while T_P decreases with the number of sub-domains.

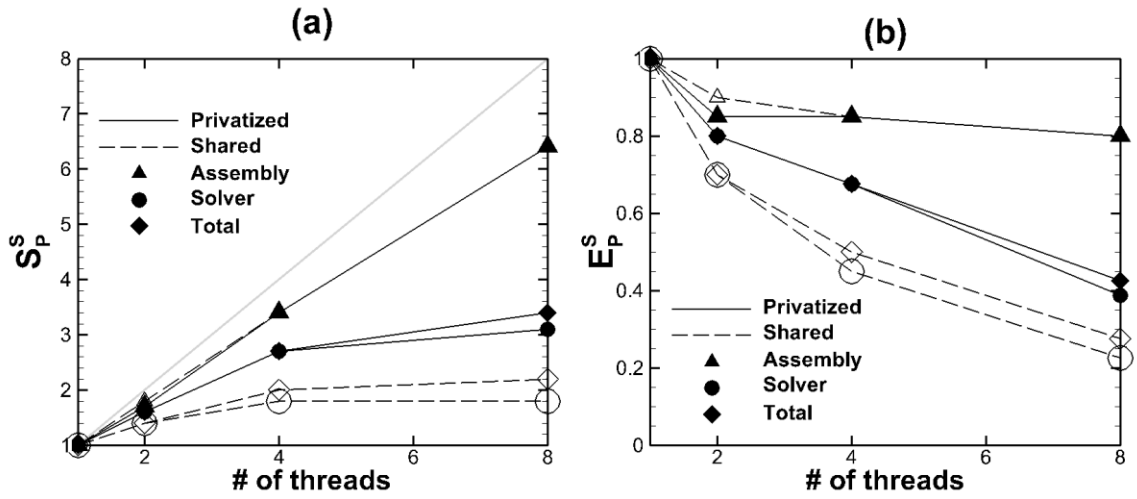

Figure 3.22 Results of speedup and parallel efficiency for variably-saturated flow simulation 1 consisting of 10^5 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.22 shows the parallel speedup and efficiency results for simulation 1 when PHGS is optimized for speed. The parallel speedup, S_P^S, for the cases with and without privatization increases as the number of threads used increases. The maximum speedup with privatization is about 6.4 ($E_8^S = 0.80$) for matrix assembly and it is 3.1 ($E_8^S = 0.38$) for the solver. The maximum speedup without privatization is about 6.4 ($E_8^S = 0.80$) for assembly and 1.8 ($E_8^S = 0.23$) for the solver. Maximum overall speedup with privatization is 3.4 ($E_8^S = 0.42$), and it is about 2.2 ($E_8^S = 0.28$) with the shared scheme.

The parallel efficiency, E_P^S, for the cases with and without privatization decreases with the increasing number of threads. However, except for matrix assembly, E_P^S for the solver and for the overall efficiency is higher with the privatization scheme being applied than without.

3.5.3.2 Results for Transient Variably-Saturated Flow Simulation 2

The overall serial (T_S) and parallel (T_P) computing times required for simulation 2 under various simulation conditions are listed in Table 3.22 when Parallel HydroGeoSphere (PHGS), compiled without optimization, is used. The computing time, T_s, obtained with natural ordering is 20402 sec, and the average T_s value with reordering is about 17169 sec. Reordering improves the efficiency of serial computing by about 16 %. For the serial computing with the reordering scheme (T_s for 2, 4, and 8 SDs), privatization slightly improves the parallel efficiency, but the difference between the cases with and without privatization is less than 5 %. In general, T_P decreases as the number of threads used increases, while T_s increases with increasing number of sub-domains.

Table 3.22 Total serial (T_S) and parallel (T_P) computing times for variably-saturated flow simulation 2 with different numbers of threads using a non-optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	20402[1)]	16800 (8821)	16753 (4667)	17063 (2875)
Shared		16854 (10331)	17621 (6458)	17921 (4486)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

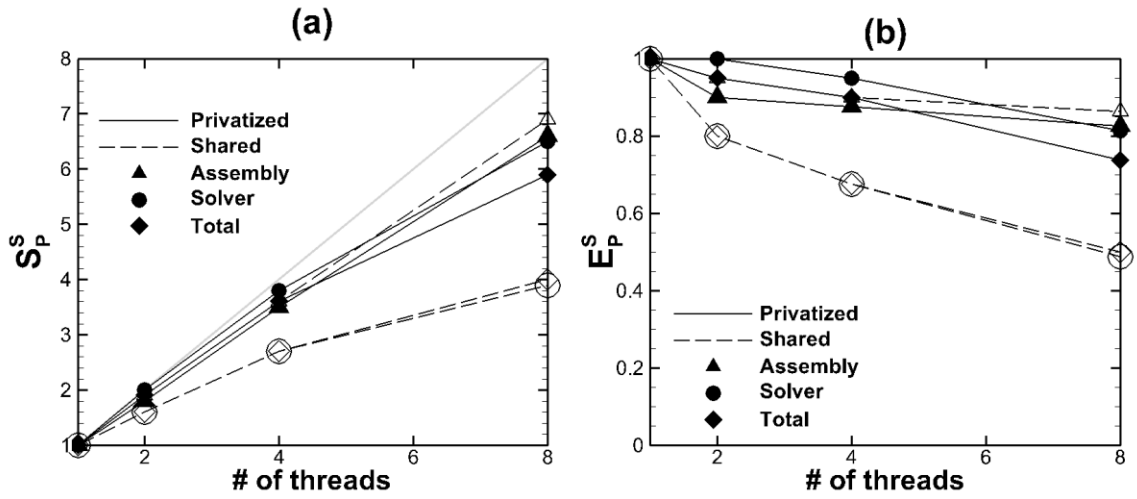

Figure 3.23 Results of speedup and parallel efficiency for variably-saturated flow simulation 2 consisting of 10^6 nodes with a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.23 illustrates parallel speedup results, S_P^S, and the parallel efficiency, E_P^S, for simulation 2 when PHGS is compiled without optimization for speed. S_P^S for the cases with and without privatization is proportional to the number of threads, but the rate of increase is higher when the privatization scheme is applied. The maximum speedup with the privatization approach is about 6.6 ($E_8^S = 0.83$) for matrix assembly and 6.5 ($E_8^S = 0.81$) for the matrix solver. Maximum speedup without privatization is about 6.9 ($E_8^S = 0.86$) for assembly, and it is 3.9 ($E_8^S = 0.49$) for the solver. The maximum overall speedup with the privatization scheme is 5.9 ($E_8^S = 0.74$), and it is 4.0 ($E_8^S = 0.50$) without privatization.

The parallel efficiency, E_P^S, decreases with an increasing number of threads. Compared to the shared scheme, E_P^S for the parallel solver is improved by the privatization scheme, and the improved computing efficiency produces an increase in the overall efficiency even though E_P^S for matrix assembly is similar for the shared and privatized cases.

Table 3.23 Total computing times, T_s and T_P, for variably-saturated flow simulation 2 with different numbers of threads using an optimized version of PHGS

Communication Scheme	T_s (T_P) in sec			
	1 SD	2 SDs	4 SDs	8 SDs
Privatized	7172[1)]	6225 (4104)	6206 (2492)	6278 (2117)
Shared		6097 (4780)	6165 (2898)	6222 (3508)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

Table 3.23 lists the computing times for the serial simulations, T_s, using natural ordering (no partitioning) and reordering with 2, 4, and 8 sub-domains when PHGS is optimized for speed. To investigate the computing efficiency of the reordering scheme, T_s values with different numbers of partitioned domains are compared. A mean T_s value for the cases with reordering scheme (2, 4, and 8 partitioned domains) is about 6199 sec, and T_s using the natural ordering scheme was about 7172 sec. As a result, the computing time with the reordering scheme is about 14 % less than that with natural.

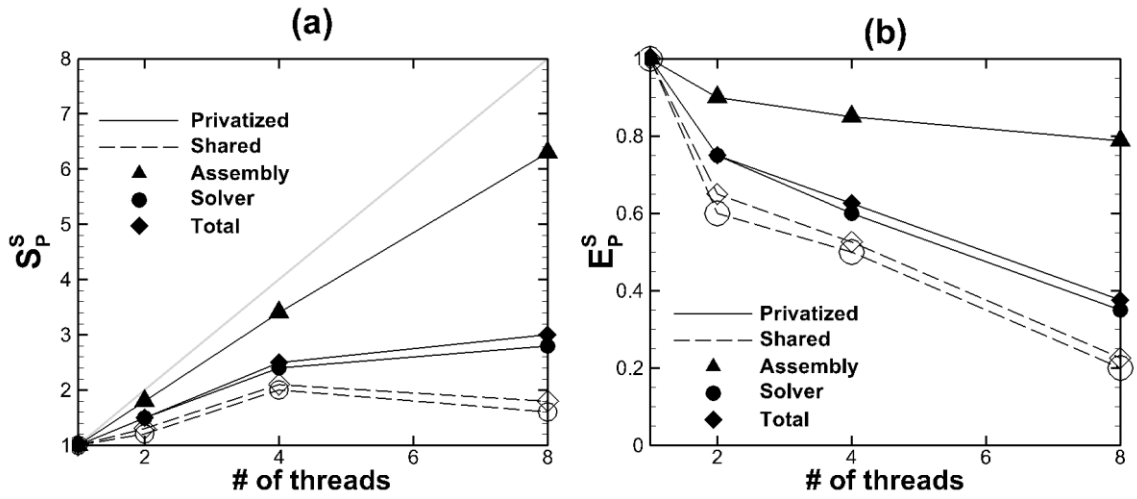

Figure 3.24 Results of speedup and parallel efficiency for variably-saturated flow simulation 2 consisting of 10^6 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.24 illustrates the parallel speedup, S_P^S, and parallel efficiency, E_P^S, with and without the privatization scheme being applied. The S_P^S value generally increases with an increasing number of threads, but it decreases when 8 threads are used without privatization. The maximum speedup with privatization is about 6.3 ($E_8^S = 0.79$) for matrix assembly and 2.8 ($E_8^S = 0.35$) for the matrix solver. Similarly, with the shared scheme, it is about 6.3 ($E_8^S = 0.79$) for assembly and 1.6 ($E_8^S = 0.20$) for the solver. The maximum overall speedup is 3.0 ($E_8^S = 0.37$) with privatization and about 1.8 ($E_8^S = 0.22$) without privatization. The overall efficiency and the efficiency of the solver with privatization are higher than those for the case without privatization.

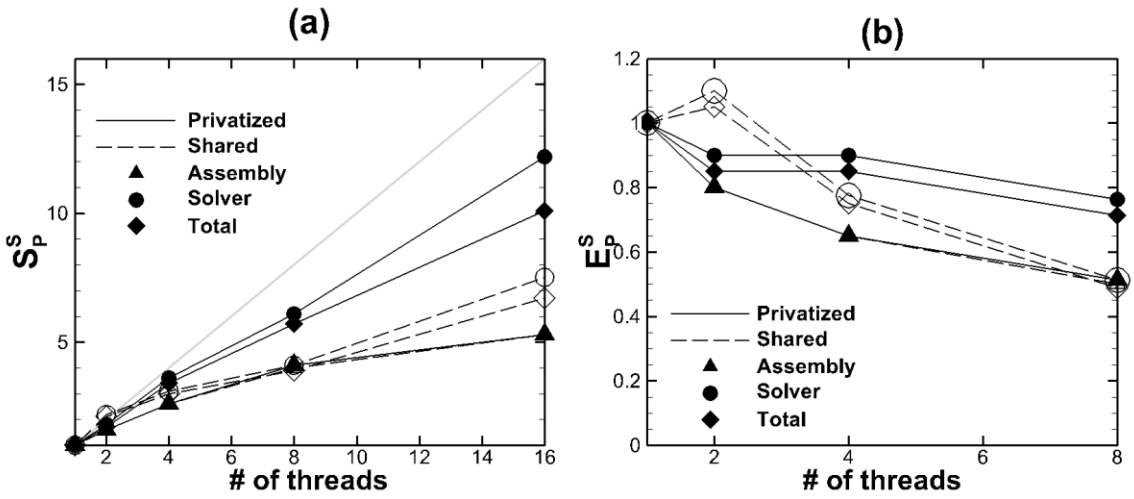

Figure 3.25 Results of speedup and parallel efficiency for variably-saturated flow simulation 3 consisting of 10^7 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Table 3.24 Total computing times, T_s and T_P for variably-saturated flow simulation 3 with different numbers of threads using an optimized version of PHGS

Communication Scheme	T_s (T_P) in sec				
	1 SD	2 SDs	4 SDs	8 SDs	16 SDs
Privatized	148928[1)]	118496 (68027)	142774 (41854)	158970 (28128)	152593 (15097)
Shared		158097 (73720)	125888 (42449)	122734 (31162)	124926 (18644)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

3.5.3.3 Results for Transient Variably-Saturated Flow Simulation 3

Computing times obtained from the serial simulations with natural ordering and with node reordering using 2, 4, 8, and 16 sub-domains are listed in Table 3.24 when PHGS is optimized for speed. The T_s value with natural ordering is about 1.5×10^5 sec, and the average of the T_s values with the reordering scheme was 1.4×10^5 sec. The cases with reordering are about 7 % faster on average than the case with natural ordering.

Figure 3.25a shows that the parallel speedup, S_P^S, with the reordering schemes increases with the number of threads applied. The S_P^S values for matrix assembly are similar with and without

privatization. In terms of S_P^S for the solver and overall computation, the privatization scheme improves S_P^S compared to the cases using the shared scheme. Figure 3.25b illustrates that the parallel efficiency, E_P^S, for the cases with and without privatization decreases with an increasing number of threads, but the E_P^S values are relatively high compared to those for simulations 1 and 2 performed on GPC. The maximum S_P^S value obtained with the privatization approach is about 5.3 ($E_{16}^S = 0.33$) for matrix assembly and 12.2 ($E_{16}^S = 0.76$) for the solver. The maximum speedup obtained without privatization is about 5.3 ($E_{16}^S = 0.33$) for assembly and 7.5 ($E_{16}^S = 0.47$) for the solver. The maximum S_P^S for overall computing is 10.1 ($E_{16}^S = 0.63$) with privatization and it is about 6.7 ($E_{16}^S = 0.42$) without privatization. Once again, this indicates that the efficiency of parallel computing is improved by applying the privatization scheme.

3.5.4 Weak Scaling Tests for Transient Variably-Saturated Flow

The simulations performed for the weak scaling tests utilize two scaling compartments (see Figure 3.2). Table 3.25 lists the variably-saturated flow simulations performed for the weak scaling tests on GPC and TCS. The number of nodes for the simulations performed on GPC ranges from 1×10^5 to 8×10^5. The simulations with the number of nodes ranging from 1×10^6 to 1.6×10^7 were performed on TCS using up to 16 threads. All the simulations were performed under the same simulation conditions as in the strong scaling tests. The PHGS model used was optimized for speed.

Table 3.25 Variably-saturated flow simulation cases for evaluating parallel efficiency in weak scaling tests

Thread	Number of nodes (nx × ny × nz)	
	GPC[1)]	TCS[2)]
1	100 × 100 × 10	100 × 1000 × 10
2	100 × 200 × 10	100 × 2000 × 10
4	100 × 400 × 10	100 × 4000 × 10
8	100 × 800 × 10	100 × 8000× 10
16	NA	100 × 16000× 10

GPC[1)] : General Purpose Cluster at SciNet/University of Toronto
TCS[2)] : Tightly Coupled System at SciNet/University of Toronto

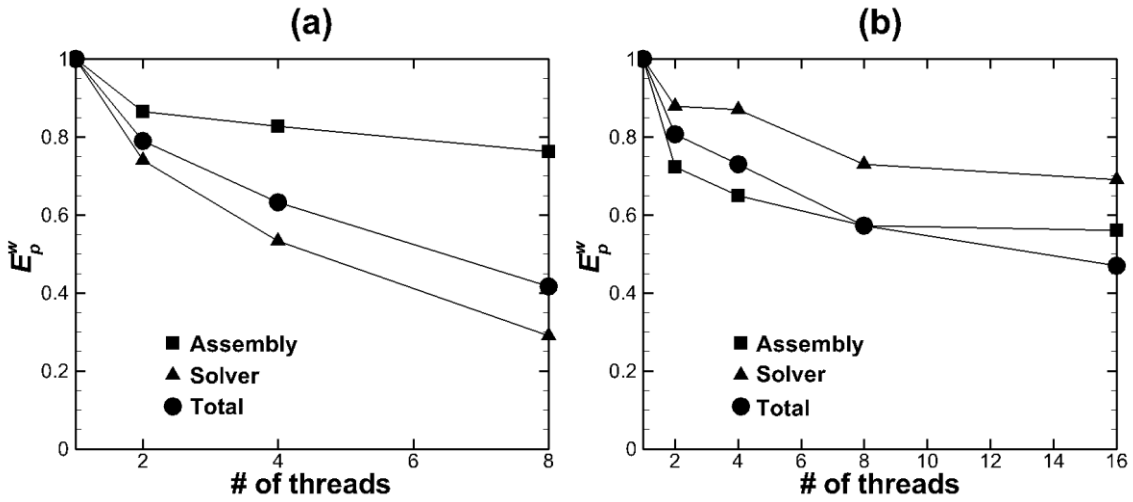

Figure 3.26 Results of weak scaling tests for variably-saturated flow performed on a) GPC and b) TCS

Figure 3.26 provides parallel efficiency results for the weak scaling tests performed on a) GPC and b) TCS. The test results show that E_p^w gradually decreases as both the number of nodes used for the simulations and the number of threads applied increases. E_8^w obtained using GPC is 0.76 for matrix assembly, 0.29 for the solver and 0.42 overall. E_p^w for the matrix assembly is higher than that for the matrix solver on GPC. On the other hand, E_{16}^w on TCS is 0.56 for assembly, 0.69 for the solver and 0.47 overall.

Similar to the results obtained from the weak scaling tests for saturated flow, the test using the smaller compartment on GPC gives a relatively low E_p^w value compared to those obtained with TCS. Specifically, E_8^w for the matrix solver is 0.29 on GPC, compared to 0.73 on TCS. E_p^w for the solver slightly decreases with an increasing number of threads on TCS, but it is higher than that for assembly.

3.6 Parallel Efficiency for Integrated Surface-Subsurface Flow Simulations

3.6.1 Simulation Description and Domain Partitioning

A rainfall-runoff simulation was adopted to evaluate the parallel computational performance for integrated surface-subsurface flow. As used in previous simulations, the domain consists of block elements but the mesh was adjusted to accommodate the sloping surface to generate runoff: the domain surface slopes at 0.05 % between x= 0 m and x= 400 m. A low-permeability slab (K_2= 1.16×10^{-7} m/s) is situated in the center of the uppermost layer (100 m × 340 m × 0.05 m in dimensions), surrounded by a more permeable medium (K_1= 1.16×10^{-5} m/s). The water table is initially located at an elevation of 4.0 m. A critical depth boundary condition is assigned to the upper line of nodes at x= 400 m (Figure 3.27). A specified flux boundary condition representing rainfall is applied to the surface domain with the rate being 5.5×10^{-6} m/s for the first 200 min of simulation time. The remaining porous medium properties for the simulation are listed in Table 3.26.

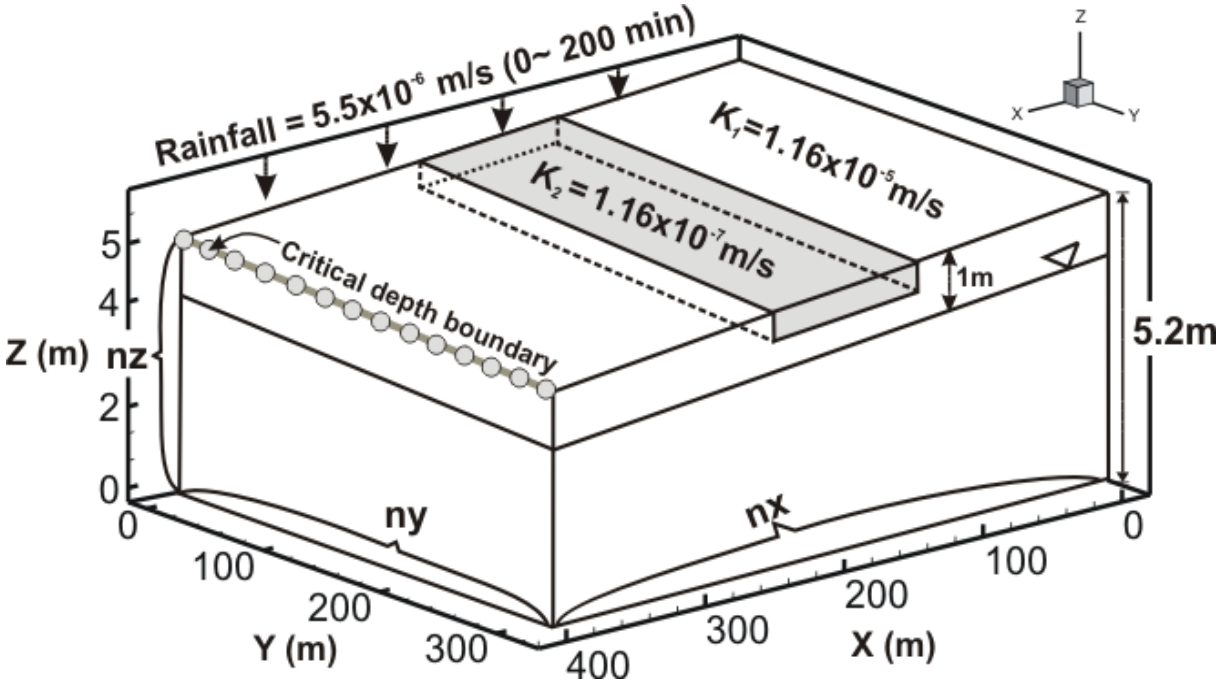

Figure 3.27 Initial and boundary conditions for integrated surface-subsurface flow simulation

Table 3.26 Prous medium properties for integrated surface-subsurface flow example.

Porous medium property	Value
Porosity [-]	0.4
Specific storage S_s [1/m]	0.0
K_1 [m/s]	1.16×10^{-5}
K_2 [m/s]	1.16×10^{-7}
Unsaturated Van Genuchten functions	
α [1/m]	1.0
β [-]	2.0
Residual saturation [-]	0.2

For the evaluation of parallel performance, Table 3.27 lists three simulation cases used for the parallel efficiency tests, where the horizontal resolution of the simulation domain varied from a maximum base case value of 4.0 m × 3.2 m to the highest resolution case with 0.1 m × 1.0 m element sizes in the x- and y-directions. Because of the limitations on memory, simulation 3 was performed on TCS, while simulations 1 and 2 were performed on GPC.

Table 3.27 Simulation cases for evaluating parallel efficiency for the integrated surface-subsurface flow example.

No	Number of node (nx × ny × nz)	Test machine
1	100 × 100 × 11	GPC[1)]
2	330 × 330 × 11	GPC
3	3300 × 330 × 11	TCS[2)]

GPC[1)] : General Purpose Cluster at SciNet/University of Toronto
TCS[2)] : Tightly Coupled System at SciNet/University of Toronto

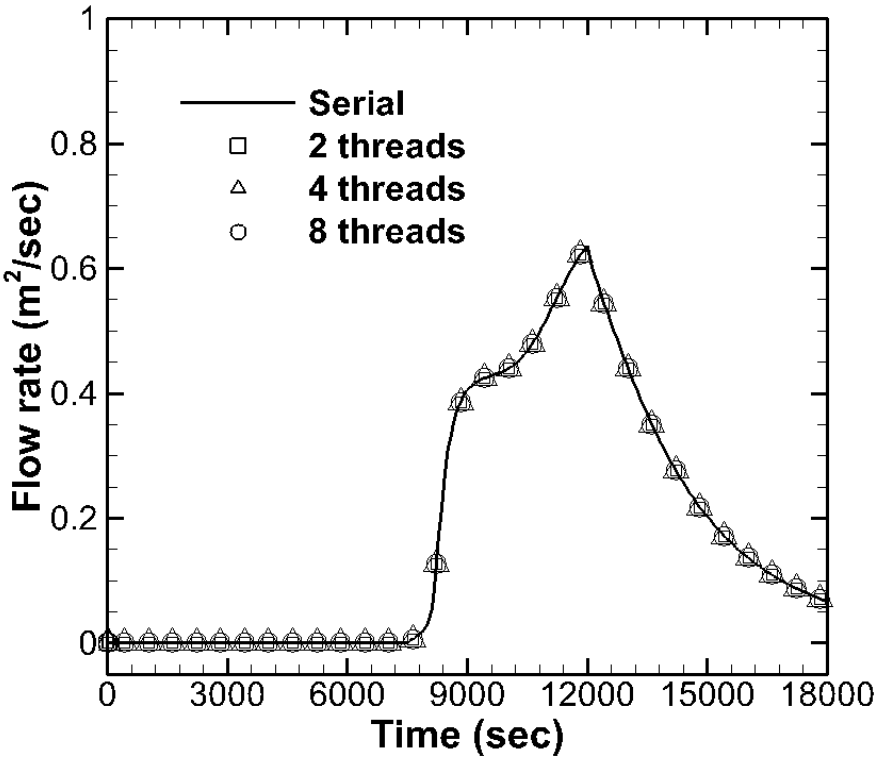

Figure 3.28 Comparison of overland flow rates for serial and parallel computing for integrated surface –subsurface flow example. The flow is the sum of the nodal flows along the critical depth boundary.

3.6.2 Consistency of Parallel Simulations

Simulation results obtained from parallel computing were compared to the results from the serial case to ensure that the numerical implementation and the solutions for the integrated surface-subsurface simulations obtained by parallel computation are accurate and consistent. Figure 3.28 provides the temporal changes in the overland flow rates at the critical boundary computed from the serial and parallel computations. The discharge at the outlet begins to increase at about 7400 sec and the flow rate gradually decreases after it peaks at 12000 sec, which is at the end of the rainfall event. The comparison of the results from the serial and parallel simulations shows that the maximum difference in the discharge at the outlet is less than 2.6×10^{-5} m^2/sec, which is negligible.

The total number of solver iterations for the serial and parallel simulations is listed in Table 3.28. The number of solver iterations in the serial simulation without node reordering is 4093 for simulation 1, 13827 for simulation 2 and 67525 for simulation 3. The number of solver iterations increases as the number of nodes used in the simulation increases. Compared to the serial simulations performed without node reordering, the total number of iterations is about 9 % less with the natural ordering scheme except for simulation 2, which is about 1 % less with the natural ordering compared to the reordering scheme.

The average total number of solver iterations required for the simulations with reordering is 4287 for simulation 1, 13918 for simulation 2 and 73703 for simulation 3. The relationship between the number of solver iterations and the number of nodes used with reordering is similar to that obtained with natural ordering. Additionally, the number of threads and the number of partitioned sub-domains influences the number of solver iterations; the number generally increases with a greater number of sub-domains. This is again partly because the sub-domain boundary nodes require additional communication between the connected sub-domains.

Table 3.28 Comparison of the number of iterations for simulation cases involving integrated surface-subsurface flow.

Thread	**Simulation 1** (4093)[1)]		**Simulation 2** (13827)[1)]		**Simulation 3** (67525)[1)]	
	Privatized	**Shared**	**Privatized**	**Shared**	**Privatized**	**Shared**
1	4303	4314	13928	13946	72801	73130
2	4389	4504	13875	13899	69769	69711
4	4576	4280	13945	13880	72860	72781
8	3968	3966	13948	13927	73187	77787
16	NA	NA	NA	NA	77500	77500

[1)]: Number of iterations in serial computing without reordering

3.6.3 Strong Scaling Tests for Integrated Surface-Subsurface Flow

3.6.3.1 Results for Integrated Surface-Subsurface Flow Simulation 1

For the case where Parallel HydroGeoSphere (PHGS) was compiled without an optimization option, Table 3.29 lists the computing times for the serial, T_S, and parallel, T_P, simulations with the natural ordering and reordering schemes. Serial simulations with reordering are based on the different numbers of partitioned sub-domains used but they are performed with a single thread, while the parallel simulations are performed with the number of threads being the same as the number of sub-domains. The T_S value with reordering does not change significantly with the number of sub-domains, but T_P decreases as the number of threads increases. The T_S value obtained using the natural ordering scheme is 1119 sec, and the mean T_S value with the reordering scheme with different numbers of partitioned sub-domains is 1142 sec. The difference in T_S between the cases with and without privatization is about 2 % and thus the effect of privatization here is insignificant for the integrated surface-subsurface flow problem. The T_P values with privatization are consistently less than those without privatization.

Table 3.29 Total computing times, T_S and T_P, for integrated surface-subsurface flow simulation 1 with different numbers of threads using an non-optimized version of PHGS

Communication Scheme	**T_S (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	1119[1)]	1147 (628)	1144 (336)	1123 (189)
Shared		1144 (670)	1158 (381)	1137 (218)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

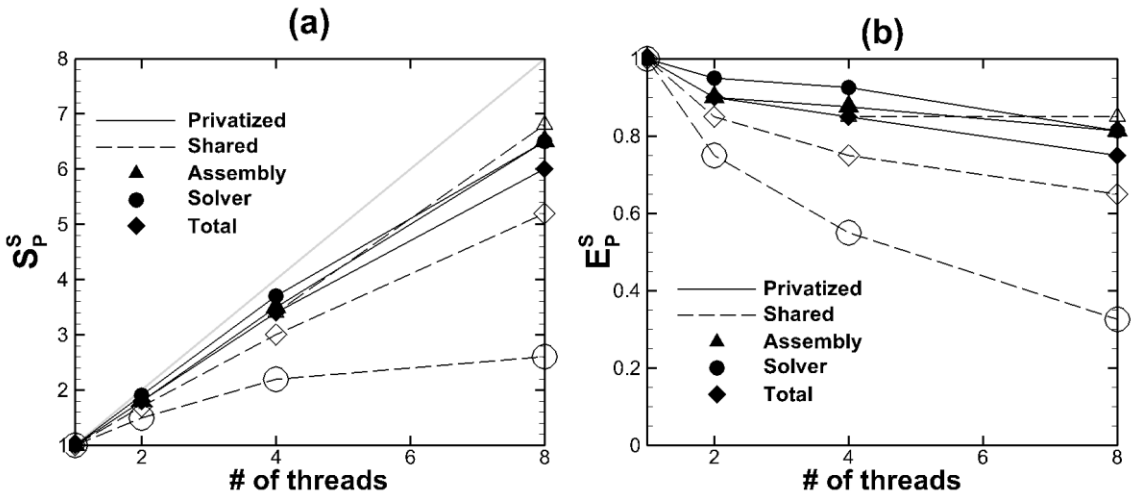

Figure 3.29 Results of speedup and parallel efficiency for integrated surface-subsurface flow simulation 1 consisting of 10^5 nodes with a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.29 shows the parallel speedup, S_P^S, and parallel efficiency, E_P^S, results for simulation 1 when PHGS was compiled without being optimized for speed. S_P^S for the cases with and without privatization increases with an increasing number of threads used. The maximum S_P^S with the privatization approach is about 6.5 ($E_8^S = 0.81$) for matrix assembly and 6.5 ($E_8^S = 0.81$) for the matrix solver. With the shared scheme, the maximum speedup is about 6.8 ($E_8^S = 0.85$) for assembly and 2.6 ($E_8^S = 0.33$) for the solver. Maximum overall S_P^S is 6.0 ($E_8^S = 0.74$) with privatization, and it is about 5.2 ($E_8^S = 0.65$) without privatization. E_P^S for both cases decreases with an increasing number of threads, but the rate of decrease in E_P^S for the solver and the overall simulation is lower with privatization. Specifically, the minimum E_P^S of the solver was 0.82 and 0.33 with and without privatization, respectively.

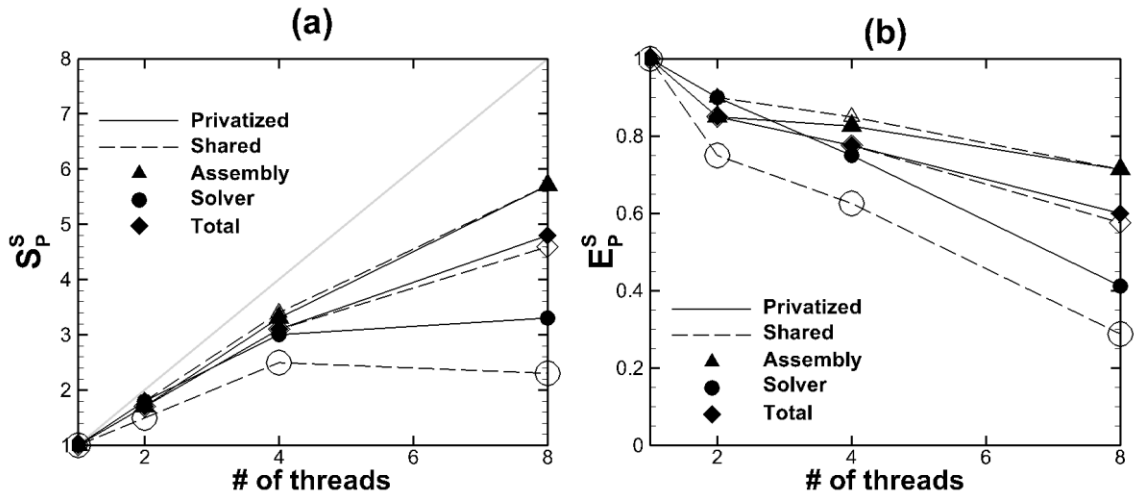

Figure 3.30 Results of speedup and parallel efficiency for integrated surface-subsurface flow simulation 1 consisting of 10^5 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Table 3.30 Total computing times, T_s and T_P, for integrated surface-subsurface simulation 1 with different numbers of threads using an optimized version of PHGS

Communication Scheme	T_s (T_P) in sec			
	1 SD	2 SDs	4 SDs	8 SDs
Privatized	445[1)]	449 (261)	450 (145)	444 (92)
Shared		451 (263)	453 (147)	447 (98)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

When PHGS is optimized for speed, the serial computing time T_s is about one third compared to the case without optimization. Overall T_s and T_P values with up to 8 sub-domains are listed in Table 3.30. T_s with natural ordering is 445 sec and it is 449 sec with the reordering scheme when averaged over the cases for the different numbers of sub-domains. The serial computing time with natural ordering is similar to the averaged computing time with reordering. There is no significant difference in T_s with and without privatization.The T_P values with privatization are also similar to those without privatization.

Figure 3.30 shows the results of the parallel speedup, S_P^S, and the parallel efficiency, E_P^S, for simulation 1 when PHGS is optimized for speed. The maximum speedup with the privatization

approach is about 5.7 ($E_8^S = 0.71$) for matrix assembly and 3.3 ($E_8^S = 0.41$) for the solver. Without privatization, the maximum speedup is about 5.7 ($E_8^S = 0.71$) for assembly and 2.3 ($E_8^S = 0.29$) for the solver. The maximum overall speedup is 4.8 ($E_8^S = 0.60$) with privatization and 4.6 ($E_8^S = 0.57$) without privatization. E_P^S for the cases with and without privatization decreases with an increasing number of threads. Among the computing tasks, matrix assembly exhibits a higher efficiency than the matrix solver, and E_P^S values for assembly are similar with and without privatization. The privatization approach improved the E_P^S values for the solver and thus enhanced the overall parallel efficiency.

3.6.3.2 Results for Integrated Surface-Subsurface Flow Simulation 2

Serial computing times, T_s, for simulation 2 are listed in Table 3.31 when PHGS is not optimized for speed. The T_s value with natural ordering is 17459 sec, and the mean T_s value with reordering is 17171 sec. The difference in computing time is less than 2 % of the mean T_s for the case with reordering. Thus, similar to simulation 1, the improvement in computing speed is also insignificant using the reordering scheme. With the privatization scheme, it is slightly faster but the difference is less than 5 %.

Table 3.31 Total computing times, T_s and T_P, for surface-subsurface simulation 1 obtained with different numbers of threads using a non-optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**			
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**
Privatized	17459[1)]	16821 (9184)	16789 (4797)	16836 (2793)
Shared		17321 (11530)	17526 (5967)	17732 (4129)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

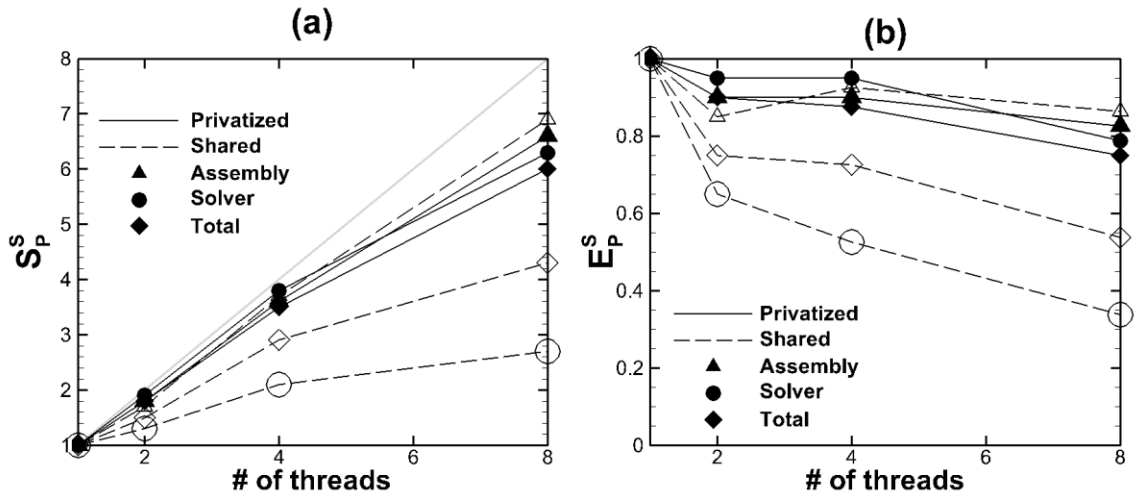

Figure 3.31 Results of speedup and parallel efficiency for integrated surface-subsurface simulation 2 consisting of 10^6 nodes with a non-optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.31 shows the results of parallel speedup, S_P^S, and parallel efficiency, E_P^S, for simulation 2. The S_P^S values for the cases with and without privatization increase as the number of threads (and hence sub-domains) increases. The increasing rates for the solver and for overall computing with privatization are higher than those without privatization. The maximum speedup with privatization is about 6.6 ($E_8^S = 0.82$) for matrix assembly and 6.3 ($E_8^S = 0.79$) for the solver, while it is about 6.9 ($E_8^S = 0.86$) for assembly and 2.7 ($E_8^S = 0.33$) for the solver without privatization. Maximum overall speedup is 6.0 ($E_8^S = 0.75$) with privatization and it is 4.3 ($E_8^S = 0.54$) without privatization.

The parallel efficiency, E_P^S, for the cases with and without privatization decreases with an increasing number of threads, but the decreasing rate in E_P^S is lower with privatization. Compared to the case without privatization, E_P^S for the matrix solver is improved with privatization, and the improved computing efficiency leads to an increase in overall efficiency.

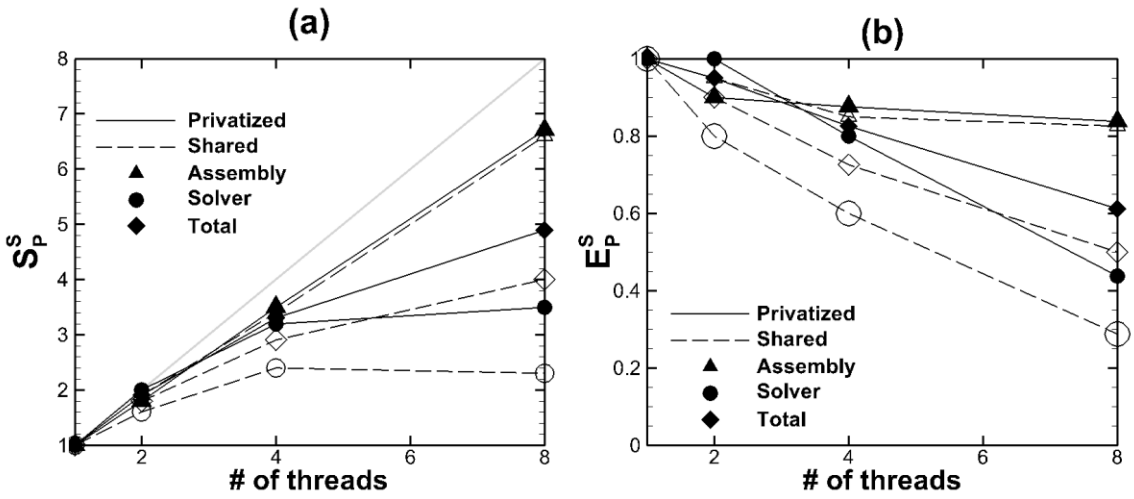

Figure 3.32 Results of speedup and parallel efficiency for integrated surface-subsurface simulation 2 consisting of 10^6 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Table 3.32 Total computing times, T_s and T_P, for integrated surface-subsurface simulation 2 with different numbers of threads using an optimized version of PHGS

Communication Scheme	T_s (T_P) in sec			
	1 SD	2 SDs	4 SDs	8 SDs
Privatized	8688[1)]	8617 (4652)	8617 (2633)	8617 (1773)
Shared		8577 (4855)	8599 (3005)	8588 (2154)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

Table 3.32 lists the computing times for the serial simulations, T_s when PHGS is compiled with optimization for speed. To investigate the computing efficiency of the reordering scheme in this case, serial computing times for simulations with natural ordering and with reordering are compared. The mean T_s value for the cases with reordering is about 8603 sec, and T_s with the natural ordering scheme is about 8688 sec. The average computing time for the serial simulations with reordering is about 1 % more than that for the simulation with natural ordering, which is negligible.

Figure 3.32 shows the parallel speedup, S_P^S, and parallel efficiency, E_P^S, for the parallel computing with and without privatization. The maximum speedup with the privatization approach is about 6.7 ($E_8^S = 0.83$) for matrix assembly and 3.5 ($E_8^S = 0.44$) for the solver, while, with the shared scheme, it is about 6.6 ($E_8^S = 0.83$) for assembly and 2.3 ($E_8^S = 0.29$) for the solver. The maximum

overall speedup with privatization is 4.9 ($E_8^S = 0.61$), and it is 4.0 ($E_8^S = 0.50$) without privatization. Parallel efficiency deceases with an increasing number of threads for the cases with and without privatization. However, the parallel efficiency for matrix assembly is higher than that for the solver. Similar to the previous simulations, the privatization scheme improves the parallel efficiency.

3.6.3.3 Results for Integrated Surface-Subsurface Flow Simulation 3

Table 3.33 lists the overall serial computing times, T_s for simulation 3 under various conditions when PHGS is optimized for speed. The serial computing time is about 4.7×10^5 sec with natural ordering and the average serial computing time is 4.8×10^5 sec with reordering. The difference in T_s values between the two cases is less than 3 %. Similar to the previous results, the reordering scheme produces an increase in the number of solver iterations, but the difference in T_s is about 8 %.

Table 3.33 Total computing times, T_s and T_P, for integrated surface-subsurface simulation 3 with different numbers of threads using an optimized version of PHGS

Communication Scheme	**T_s (T_P) in sec**				
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**	**16 SDs**
Privatized	4.7×10^5 [1)]	4.9×10^5 (2.6×10^5)	5.0×10^5 (1.6×10^5)	5.1×10^5 (8.5×10^4)	5.1×10^5 (6.0×10^4)
Shared		4.5×10^5 (2.9×10^5)	4.5×10^5 (1.9×10^5)	4.7×10^5 (1.2×10^5)	4.8×10^5 (8.5×10^4)

[1)]: Total computing time obtained by serial computing with the natural ordering algorithm

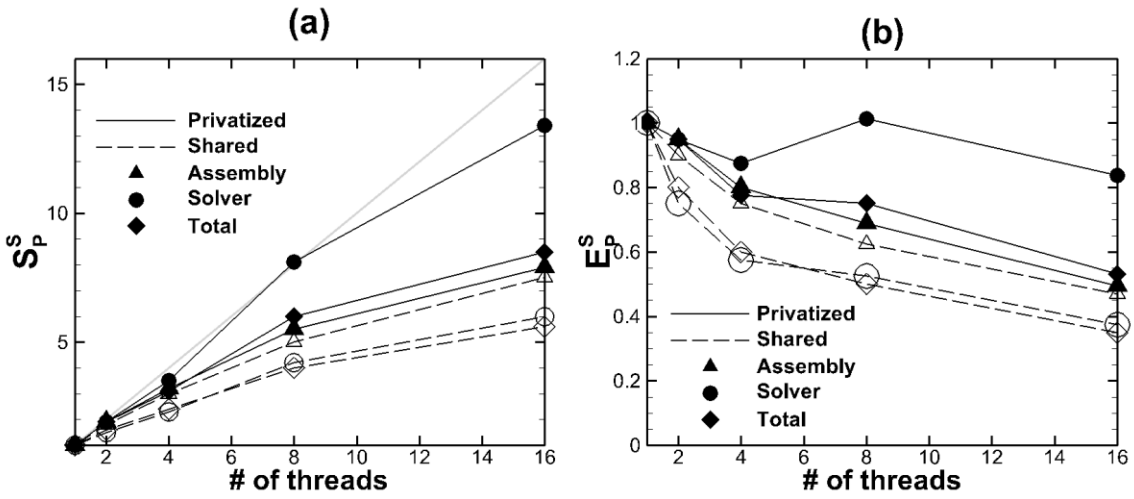

Figure 3.33 Results of speedup and parallel efficiency for integrated surface-subsurface flow simulation 3 consisting of 10^7 nodes with an optimized version of PHGS: a) parallel speedup, b) parallel efficiency

Figure 3.33 illustrates the parallel speedup (S_P^S in Figure 3.33a) and the parallel efficiency (E_P^S in Figure 3.33b) results for simulations 3. With the privatization approach, the maximum speedup is about 7.9 ($E_{16}^S = 0.49$) for matrix assembly and 13.4 ($E_{16}^S = 0.84$) for the matrix solver. Without privatization, the maximum speedup is about 7.5 ($E_{16}^S = 0.47$) for assembly and 6.0 ($E_{16}^S = 0.37$) for the solver. The maximum overall speedup with privatization is 8.5 ($E_{16}^S = 0.53$), and it is 5.6 ($E_{16}^S = 0.35$) without privatization.

Based on the evaluation results, TCS achieves greater parallel efficiency for the matrix solution with privatization than it does for matrix assembly. TCS has tightly coupled L1 and L2 caches, which improve cache efficiency for each thread. The privatization scheme applied to the matrix solver reduces competition between the threads when the threads access memory locations. Both cache efficiency and privatization can improve the speedup of the matrix solver. When compared to the parallel efficiency of the matrix solver, matrix assembly has a relatively low parallel efficiency because the privatization scheme was not applied.

3.6.4 Weak Scaling Tests for Integrated Surface-Subsurface Flow

Two weak scaling tests were performed where the number of nodes for each axis is the same as those used in the previous simulations for saturated and variably-saturated flow. Tests were performed on GPC for the problem with the smaller number of nodes and using a maximum of 8 threads and on TCS for the problem with the largest number of nodes and using a maximum 16 threads. The number of nodes and the parallel machines applied for the weak scaling tests are listed in Table 3.34. The initial and boundary conditions for the simulations are the same as those assigned for the strong scaling tests.

Table 3.34 Weak scaling test cases for evaluating parallel efficiency for integrated surface-subsurface flow

Number of threads	Number of nodes (nx × ny × nz)	
	GPC[1)]	TCS[2)]
1	100 × 100 × 11	100 × 1000 × 11
2	100 × 200 × 11	100 × 2000 × 11
4	100 × 400 × 11	100 × 4000 × 11
8	100 × 800 × 11	100 × 8000 × 11
16	NA	100 × 16000× 11

GPC[1)] : General Purpose Cluster at SciNet/University of Toronto
TCS[2)] : Tightly Coupled System at SciNet/University of Toronto

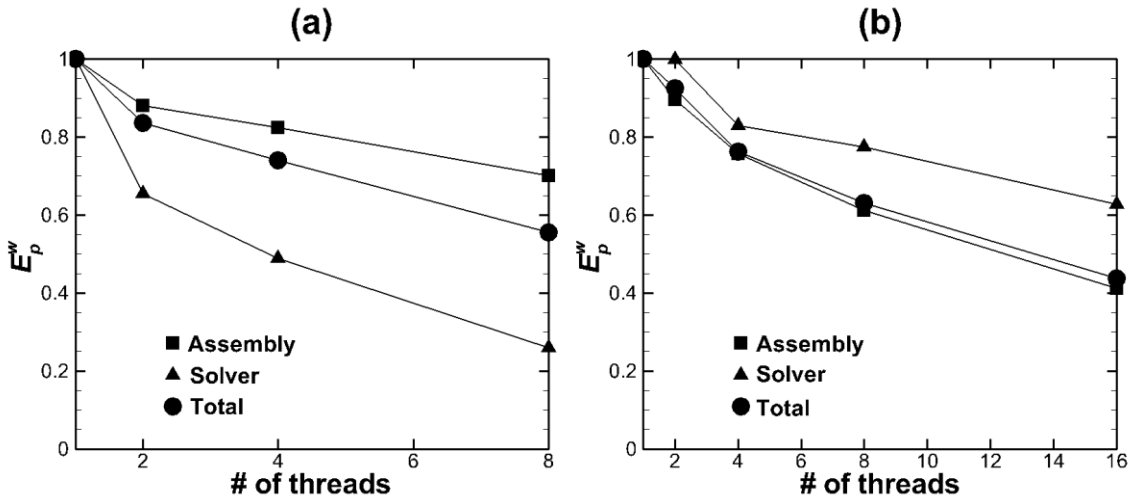

Figure 3.34 Results of weak scaling tests for integrated surface-subsurface flow performed on a) GPC and b) TCS

The results of the weak scaling tests are provided in Figure 3.34. For the smallest test problem (Figure 3.34a), the parallel efficiency for matrix assembly varies from 0.88 (=E_2^w) to 0.70 (= E_8^w), and from 0.65 (=E_2^w) to 0.26 (= E_8^w) for the solver. Overall parallel efficiency decreases as the number of threads increases: E_8^w for overall computing is 0.55. E_p^w for assembly is relatively high compared to that of the solver due to the low data dependency in assemblying the coefficient matrix. On the other hand, for the large problem, E_p^w for the solver is relatively high compared to that for assembly, which is similar to the results obtained in the previous subsurface simulations. Specifically, E_2^w for the solver and for assembly is close to 1.0 and 0.90, respectively. The parallel efficiency decreases to 0.63 for the solver and to 0.41 for assembly when 16 threads are used. Interestingly, based on the performance obtained on TCS, the parallel efficiency for the solver consistently remains higher than that for assembly. In contrast, a relatively high parallel efficiency for matrix assembly is observed for the performance on GPC. The platforms used for the parallel efficiency tests are different from two perspectives: hardware and software. In terms of software, parallel HydroGeoSphere was compiled with Intel Fortran 10.1 for GPC and xlf Fortran for TCS. For the hardware, TCS uses multilevel caches which reduce the average latency of memory access time. GPC has a shared L3 cache which has a lower access speed than L1 and L2 caches, thus causing a lower parallel efficiency for matrix solver than those obtained from TCS.

3.7 Parallel Efficiency for a Large-Scale Real-World Simulation

In order to evaluate the applicability of Parallel HydroGeoSphere, the accuracy, robustness, and scalability are investigated for a large-scale real-world application that includes conditions such as a highly heterogeneous hydraulic conductivity field, various sizes of unstructured meshes, drastic topographical changes, and various boundary conditions. For the performance evaluation, a continental-scale integrated groundwater-surface water flow simulation was adopted. A brief model description and simulation results relevant to the validity and efficiency of the model are provided in the following sections.

3.7.1 Model Description

The simulation domain consists of two interacting flow regimes, the 2-D surface and the 3-D subsurface, between which water is exchanged. The domain covers the Canadian land mass and both the unsaturated and saturated zones are included in the subsurface. Prism and triangle elements are used to discretize the subsurface and surface domains, respectively. Figure 3.35 illustrates the boundary conditions applied to the surface and subsurface domains. A critical depth boundary condition is used around the entire surface domain, and a specified flux condition is assigned on the surface to represent non-uniform net precipitation computed by the Canadian Regional Climate Model (CRCM). A specified head boundary condition is assigned along the ocean boundary in the subsurface. The target simulation time is set to be 5.0×10^5 years in order to arrive at steady flow conditions.

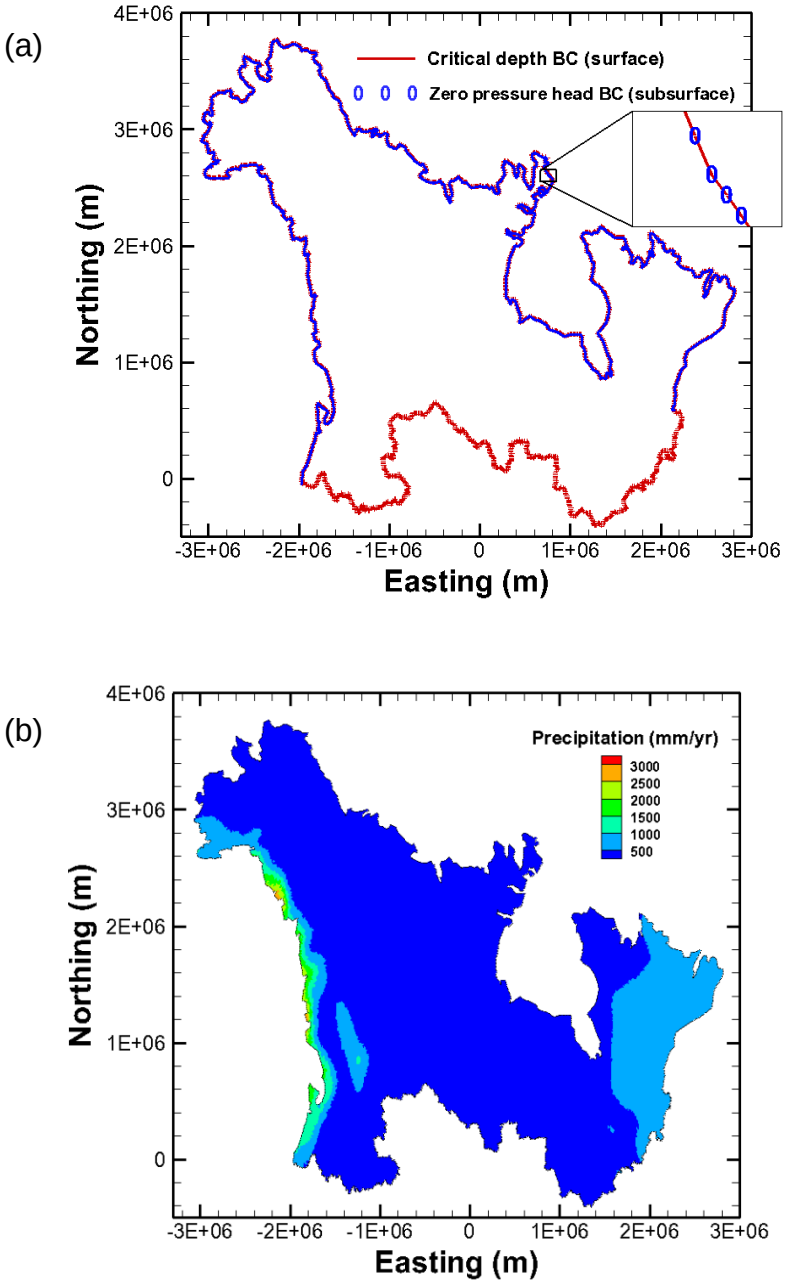

Figure 3.35 Boundary conditions assigned to the simulation domain: (a) critical depth and zero pressure head specified boundary conditions; (b) distribution of precipitation

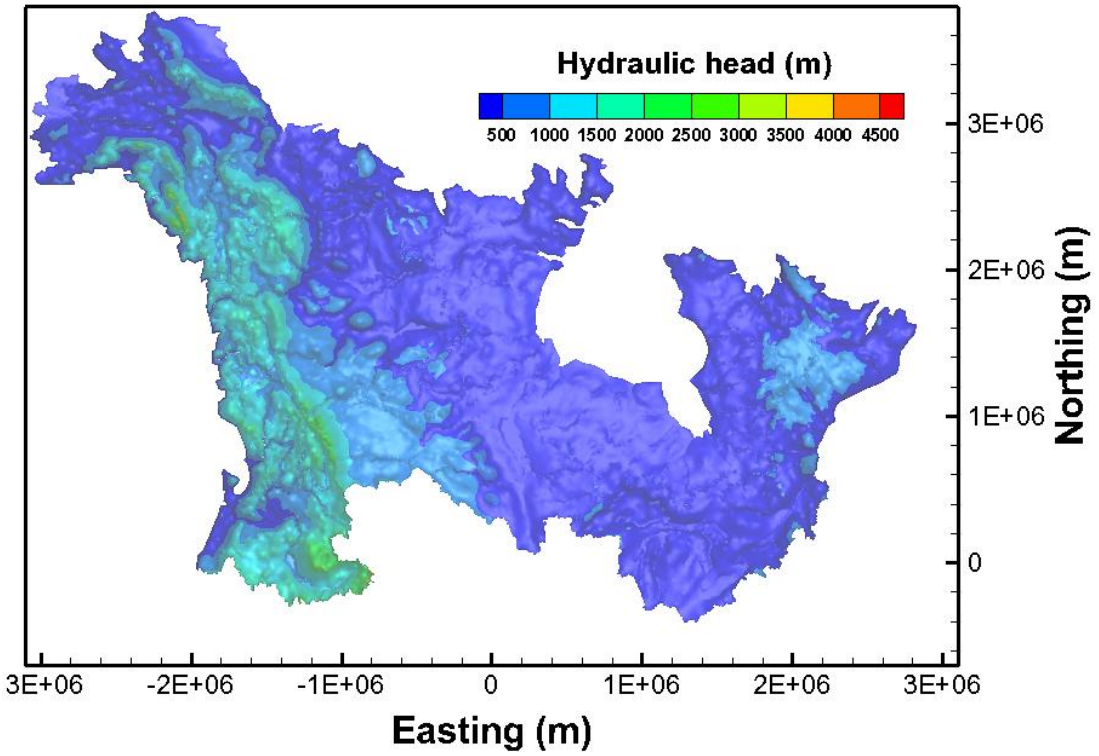

Figure 3.36 Hydraulic head distribution at pseudo-steady state for the large-scale real-world simulation.

3.7.2 Evaluation of Accuracy and Robustness

Figure 3.36 shows the simulated total hydraulic head distribution at the top of the subsurface domain at time 5.0×10^5 years. The results show that the head distribution is strongly influenced by the topography and hydraulic heads becomes higher at higher elevations. Specifically, in western Canada, where the Rocky Mountains are located, hydraulic heads range from 2000 m up to 4500 m, while in the Prairie region in central Canada, head values are relatively uniform and less than 1000 m. In other areas of high elevation such as the Laurentians and Cape Brenton Highlands, and parts of Labrador, the heads are also relatively high.

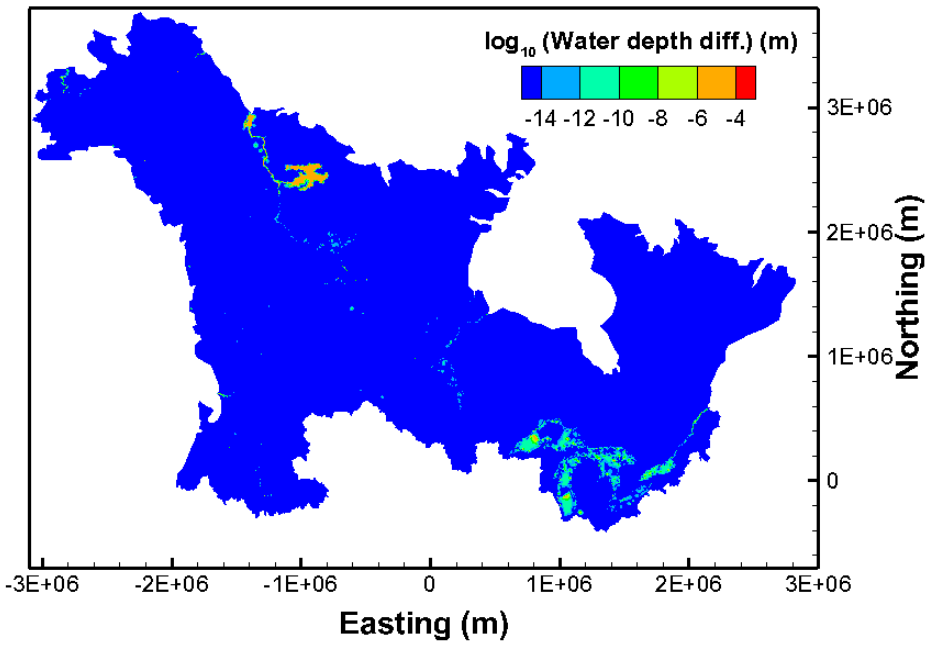

Figure 3.37 Result differences between serial and parallel simulations for large-scale real-world simulation.

Figure 3.37 illustrates the differences in simulated water depths over the land surface between the serial and parallel simulations: the differences are between one serial simulation and the average of four parallel simulations performed using 2, 4, 8, and 16 threads. The differences over most of the domain are less than 10^{-10} m except for the areas having relatively deep waterbodies such as major lakes. In Figure 3.37, the maximum difference is 10^{-4} m in the region of Great Bear Lake, located in north-west Canada. Along the Mackenzie River and the edges of Lake Michigan and Lake Ontario, the difference is in the order of 10^{-9} m. Based on this comparison, the parallel simulation results are essentially the same as those for the serial simulation.

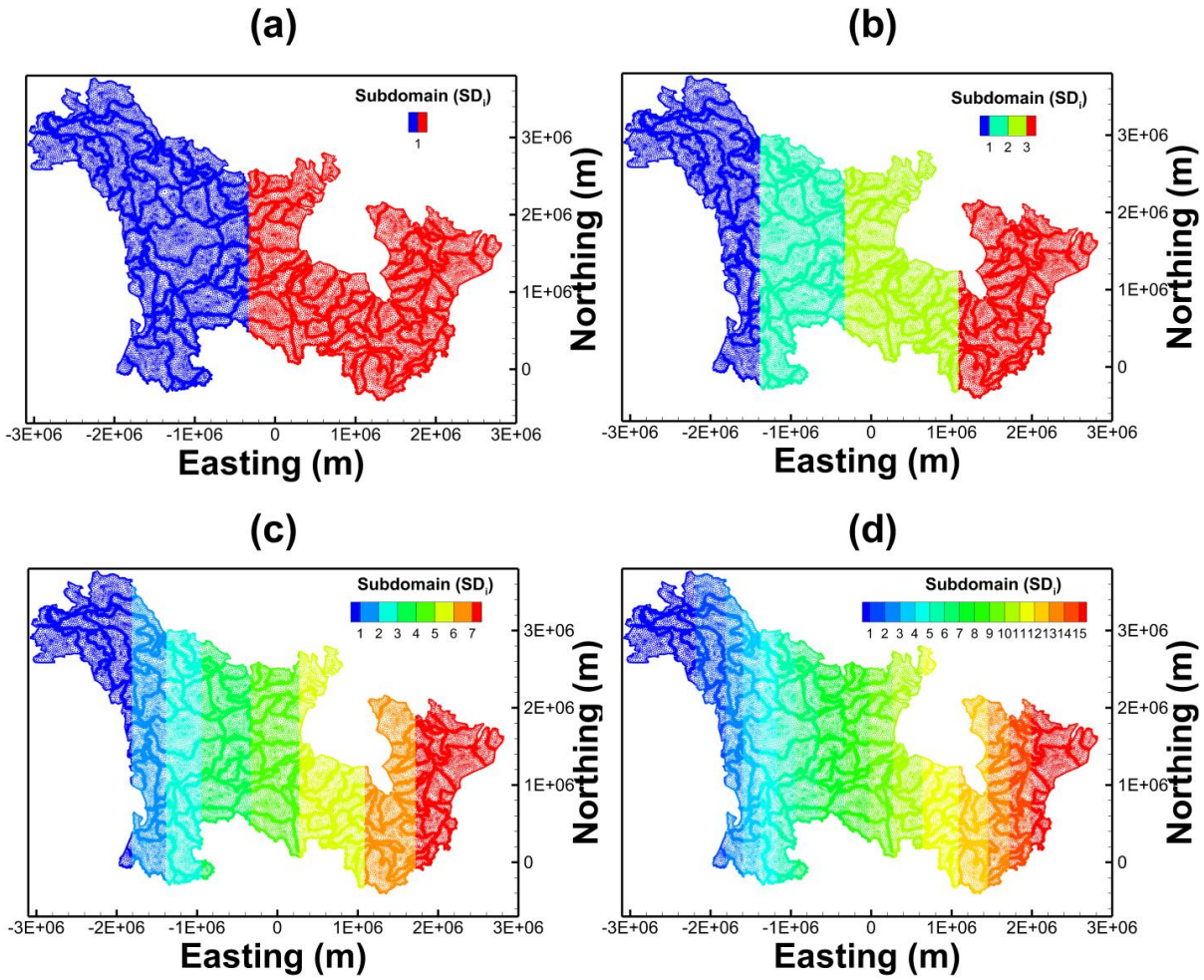

Figure 3.38 Domain partitioning for: (a) 2 threads; (b) 4 threads; (c) 8 threads; (d) 16 threads

3.7.3 Parallel Performance

By performing the parallel simulation with various numbers of threads, the parallel efficiency and speedup are evaluated based on a reference value obtained from a serial simulation. For the best efficiency, the simulation domain needs to be evenly divided according to the number of threads applied and thus the size of sub-domains generated through partitioning needs to be investigated. Figure 3.38 illustrates the partitioning of the domain into various numbers of sub-domains according to the number of threads applied. The number of nodes for each sub-domain is equally distributed to minimize the parallel overhead caused by the workload imbalance. To reduce communication traffic among threads, the separators are intended to dissect the simulation domain parallel to the *y* axis,

along which direction the smaller number of nodes are found. In the partitioning scheme applied for the simulations, two sub-domains share sub-domain boundary nodes and thus, the number of separators is 1 for two threads (Figure 3.38a), 3 for four threads (Figure 3.38b), 7 for eight threads (Figure 3.38c), and 15 for sixteen threads (Figure 3.38d).

Table 3.35 lists the number of sub-domains and the number of sub-domain boundary nodes for the partitioning shown in Figure 3.38. The total number of nodes used for the simulations is 994143, which is the sum of all the nodes for the sub-domains and sub-domain boundaries. A group of nodes located inside a sub-domain has data dependency among themselves only and thus, the solution for an individual sub-domain does not require communication with other threads. The number of nodes for the sub-domains is 497080 for two threads, 248540 for four threads, 124270 for eight threads, and 62135 for sixteen threads. The maximum difference in the number of nodes between the partitioned sub-domain is less than 10^{-5} % and thus the overall simulation domain is considered to be evenly partitioned for the parallel computations.

Parallel simulations were performed on GPC and TCS, and the total number of solver iterations were compared for the parallel simulations with different numbers of sub-domains. The number of solver iterations is an indicator of the computing efficiency because it is directly related to the number of floating point operations involved. Table 3.36 lists the number of solver iterations required for the serial and parallel simulations. The number of solver iterations for the parallel simulations increases with the number of threads used (the number of partitioned sub-domains). This is because the number of boundary nodes is proportional to the number of threads, causing a wider bandwidth of the coefficient matrix. Compared to the serial simulation with reordering, the increase in the number of solver iterations is about 8 % with 8 threads on GPC and about 18 % with 16 threads on TCS. The number of iterations with node reordering was less than 57 % of that obtained using natural ordering without the domain partitioning (i.e., the serial case). This difference is significant in terms of the number of solver iterations between the simulations with and without reordering. Based on these results, it is concluded that the node reordering plays a pivotal role in reducing the number of iterations for these types of simulations.

Table 3.35 Number of sub-domain and boundary nodes for parallel computing for large-scale real-world simulation.

Number of threads	Number of inner nodes (number of boundary nodes) in a sub-domain	
2	497080 (none),	494445 (2618)
4	248540 (none), 245922 (2618),	245293 (3247) 246670 (1853)
8	124270 (none), 121023 (3247), 121652 (2618), 122417(1853),	121142 (3128) 121533 (2737) 121482(2788) 121856(2397)
16	62135 (none), 59007 (3118), 58888 (3247), 59398 (2737), 59517 (2618), 59347 (2788), 60282 (1853), 59738 (2397),	60588 (1547) 58310 (3825) 58412 (3723) 59347 (2788) 59738 (2397) 60316 (1819) 59160 (2975) 59857 (2261)

Table 3.36 Comparison of the number of iterations for large-scale real-world simulation cases

Thread	GPC (2334672)[1)]	TCS (2334672)[1)]
1	934007[2)]	939946[2)]
2	996713	949367
4	987514	956725
8	1011091	1002539
16	NA	1113159

[1)] Number of iterations obtained by a serial on GPC with the natural ordering algorithm

[2)] Mean iteration numbers obtained from serial jobs, which are computed by parallel algorithms.

Table 3.37 lists the overall serial computing times, T_s, for the large-scale real-world simulations performed on GPC and TCS. The serial computing time, T_s, obtained with natural ordering is about 6.5×10^5 sec for GPC and 7.5×10^5 sec for TCS. TCS shows relatively low computing efficiency compared to GPC for the serial computing with natural ordering. However, the T_s value with reordering ranges from 4.1×10^5 to 4.3×10^5 sec for GPC and that for TCS ranges from 4.1×10^5 to 4.3×10^5 sec. The serial computations with reordering for TCS are almost the same as those for GPC. Similarly, the parallel computing times, T_P, for GPC and TCS are also similar to each other.

Table 3.37 Total computing times, T_s and T_P, for large-scale real-world simulation cases with different computing machines.

Computing machine	T_s (T_P) **in sec**				
	1 SD	**2 SDs**	**4 SDs**	**8 SDs**	**16 SDs**
GPC	6.5×10^5 [1)]	4.3×10^5 (2.4×10^5)	4.1×10^5 (1.3×10^5)	4.2×10^5 (9.4×10^4)	NA
TCS	6.5×10^5 [1)]	4.1×10^5 (2.4×10^5)	4.2×10^5 (1.3×10^5)	4.3×10^5 (9.4×10^4)	4.3×10^5 (6.6×10^4)

[1)]: Total computing time obtained by serial computing on GPC with the natural ordering algorithm

The parallel performance with the various number of threads is evaluated using Equations (3) and (4). Based on the profiling results from the serial simulation, most of computing time for the integrated surface-subsurface flow simulation with an unstructured mesh is spent amongst three computing tasks: matrix assembling (60 %), preconditioning (2 %) and matrix solving (25 %). The numbers in the brackets are the percentage of the computing time taken by each task compared to the total simulation time. In this section, two major computing tasks, matrix assembly and matrix solution are discussed in terms of parallel efficiency.

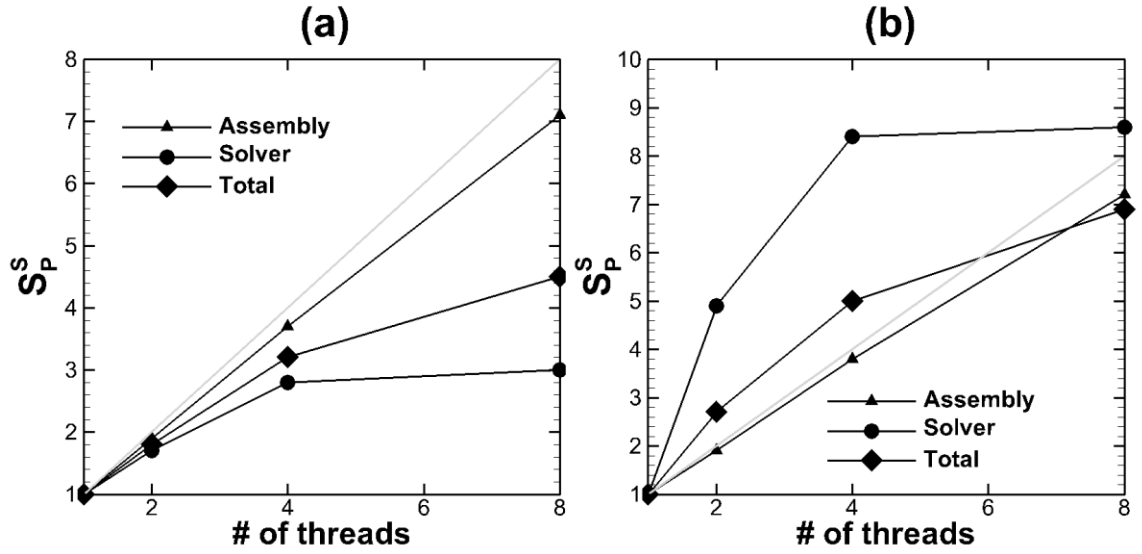

Figure 3.39 Results of speedup for large-scale real-world simulations performed on GPC: a) speedup results based on parallel algorithms and b) based on serial algorithms.

Figure 3.39 shows the parallel speedup results for the various simulations. The speedup, S_P^S, increases as the number of threads increases. S_P^S for the solver is less compared to S_P^S for matrix assembly with 8 threads applied. Figure 3.39a provides S_P^S values obtained with different numbers of threads based on the serial computations with domain partitioning scheme. The maximum parallel speedup obtained by applying 8 threads, S_8^S, is 7.1 ($E_8^S = 0.89$) for matrix assembly, 3.0 ($E_8^S = 0.37$) for the solver, and 4.5 ($E_8^S = 0.56$) overall. Because the efficiency E_8^S for assembly is relatively high compared to that of the solver, and a significant portion of the computing time is spent for matrix assembly (i.e., 60 % of total computing time), the overall efficiency is improved reasonably well even though E_8^S for the solver is relatively low.

Figure 3.39b shows the results for S_P^S compared to that for the serial simulation performed with natural ordering. Compared to the serial simulations with domain partitioning, the computing time increases by about 1.6 times. The speedup with 8 threads S_8^S for matrix assembly is 7.2 ($E_8^S = 0.90$). Interestingly, the maximum S_P^S for the solver is 8.6 ($E_8^S = 1.08$), which is higher than linear scalability (gray line). The high scalability of the solver contributed to this behaviour for the overall computation when 2 or 4 threads are applied. The overall parallel efficiency is 1.37 with two threads and 1.26 with four threads.

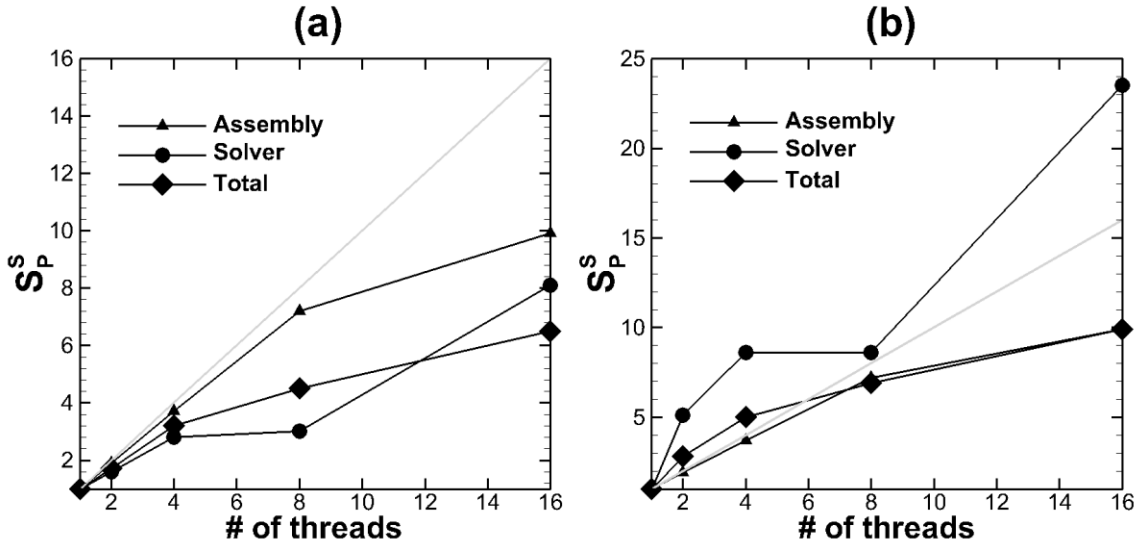

Figure 3.40 Results of speedup for large-scale real-world simulations performed on TCS: speedup with a) reordering and b) natural ordering schemes.

The parallel speedup was also evaluated on TCS (Figure 3.40). Similar to the results obtained with GPC, the speedup S_P^S was evaluated compared to the serial simulations a) with the reordering scheme and b) with natural ordering (Figure 3.40). The speedup S_P^S for matrix assembly has a close-to-linear relationship with the number of threads and the maximum S_P^S obtained is 9.9 ($E_{16}^S = 0.62$). Although S_P^S for the solver also increases with the number of threads, its rate of increase with an increasing number of threads is lower than that for assembly. The maximum S_P^S for the solver is 8.1 ($E_{16}^S = 0.51$). In terms of overall computing costs, the maximum S_P^S is 6.5 ($E_{16}^S = 0.41$), which can be compared to S_8^S of 4.5 ($E_8^S = 0.57$).

There is a factor that contributed to the decrease in the overall speedup despite the relatively high parallel efficiency for matrix assembly and matrix solution. The simulation requires a significant portion of the computing time be spent during matrix preconditioning. Specifically, the matrix preconditioning task, which is not parallelized, requires about 32 % of the total computing time due to an increase in Newton-Raphson iterations.

Figure 3.40b shows the parallel speedup relative to the serial simulation with the natural ordering scheme. For the results on TCS, there is a significant performance difference in S_P^S for the solver as well as for the overall S_P^S because the serial simulation with natural ordering spends more computing time during the matrix solution task. Specifically, the iterative solver requires about 54 %

of the computing time and assembly requires about 45 % of the total computing time. The serial simulation with natural ordering requires more than two times as many solver iterations compared to the case with node reordering.

3.8 Discussion

3.8.1 Assembly- Versus Solver-Intensive Simulations

Major numerical tasks involved in distributed parameter hydrologic modeling include matrix assembly, matrix solution, matrix preconditioning and other tasks. Among these major tasks, matrix assembly and solution were two main parallelization targets for the HydroGeoSphere model. Based on the computing costs required for the various different numerical tasks, hydrologic simulations can be categorized as solver-intensive, assembly-intensive, preconditioning-intensive or intermediate. For example, a simulation can be classified as solver-intensive if the cost of the matrix solution is more than one half of the total computing cost and thus the following relation holds:

$$\frac{T_{solver}}{T_{total}} \geq 0.5 \tag{3.4}$$

where T_{solver} is the computing time taken for the iterative matrix solver and T_{total} is the total computing time taken for the simulation. If none of the three major tasks requires more than one half of the total computing time, it may be categorized as an intermediate case:

$$\frac{T_{solver}}{T_{total}} < 0.5 \ \cap \ \frac{T_{mtrix\ assembly}}{T_{total}} < 0.5 \ \cap \ \frac{T_{precond.+others}}{T_{total}} < 0.5 \tag{3.5}$$

where $T_{mtrix\ assembly}$ is the computing time taken for the matrix assembly part and $T_{\text{precond.+others}}$ is the total computing time taken for the preconditioning and other processes such as pre- and post-processes. The intermediate type will plot in the the centre of the ternary diagram. To characterize the simulations performed to assess the parallel efficiency, the serial simulations for steady-state saturated flow (open circles), transient solute transport (closed circles), variably-saturated flow (open triangles), and integrated surface-subsurface flow with a structured mesh (open squares) and with unstructured mesh (closed squares) were plotted on the ternary diagram consisting of axes for the solver, assembly, and the precondition and others (Figure 3.41).

Transient solute transport simulations were distributed over a large spectrum of categories as illustrated in Figure 3.41: they range from assembly-intensive with 10^5 mesh nodes to solver-intensive with 10^7 nodes. At each transport time step, the average number of solver iterations was about 14 for a mesh with 10^5 nodes, 51 for 10^6 nodes and 86 for 10^7 nodes. The simulations consisting of 10^5 nodes and 10^6 nodes were performed with heterogeneous hydraulic conductivity

fields, while that consisting of 10^7 nodes was performed with a homogeneous conductivity field, which should take a smaller number of iterations. The increasing number of solver iterations with an increasing number of grid nodes shifts the simulation type to become solver-intensive.

For variably-saturated flow, most of the simulations performed were of the solver-intensive type. This is demonstrated by the fact that the mean number of solver iterations at each Newton-Raphson iteration step was about 121 for 10^5 nodes, 252 for 10^6 nodes and 302 for 10^7 nodes. The mean numbers of solver iterations was relatively high compared to those obtained from the solute transport simulations, which is a linear problem.

In contrast to the variably-saturated flow simulations, the integrated surface-subsurface flow simulations were found to be assembly-intensive but they became solver-intensive as the number of grid nodes increased. The mean numbers of solver iterations for this case are smaller than those obtained from the variably-saturated flow simulations. Specifically, the mean number of iteration at each Newton-Raphson iteration step for integrated surface-subsurface flow is about 12 for 10^5 nodes, 34 for 10^6 nodes, and 51 for 10^7. For integrated surface-subsurface flow simulations employing an unstructured mesh consisting of about 10^6 nodes, the simulation performance was similar to that obtained using 10^7 nodes, which implies that the mean numbers of iterations for both cases are similar.

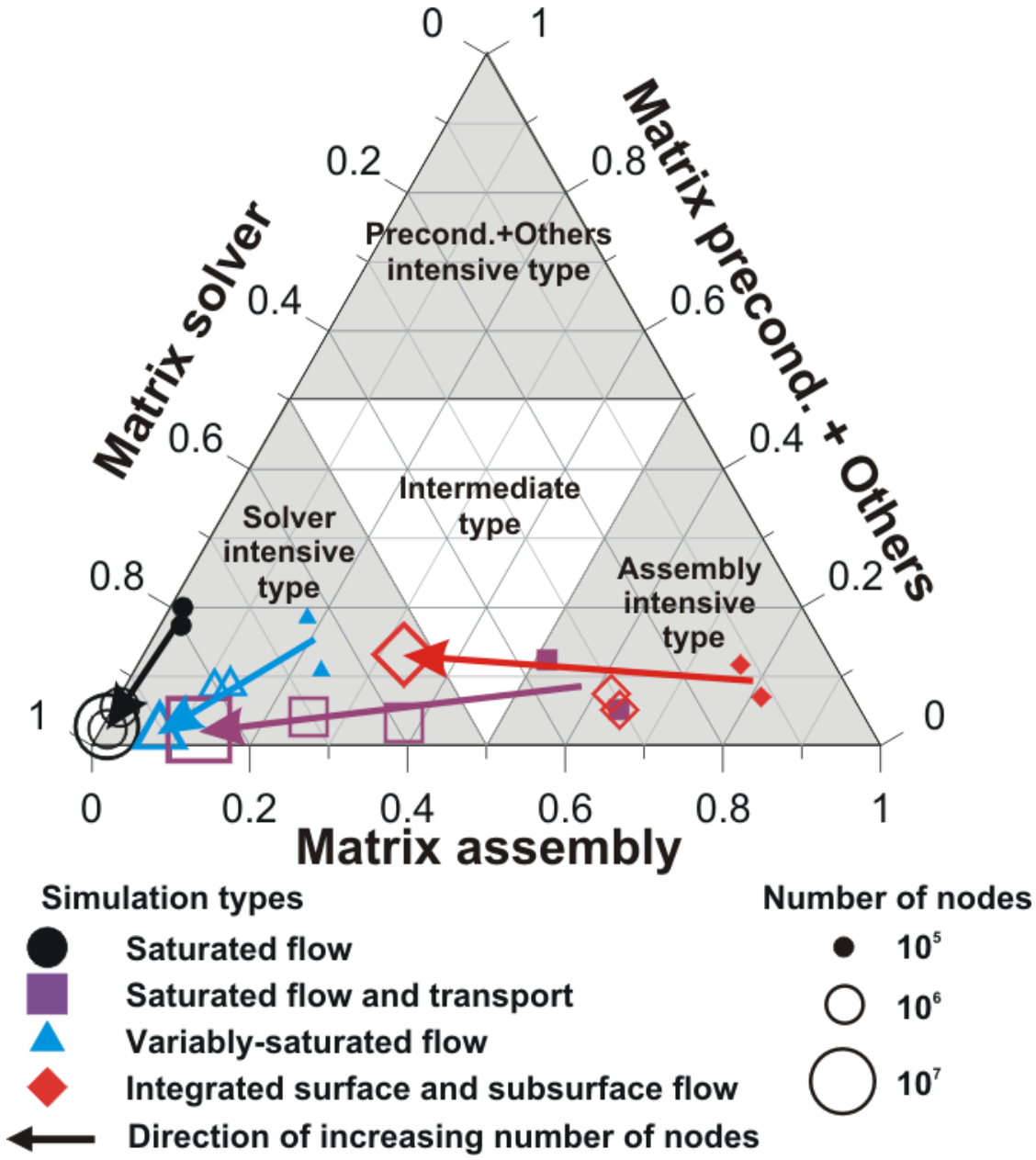

Figure 3.41 Characteristic relative computing intensities for different code segments for various hydrological problems.

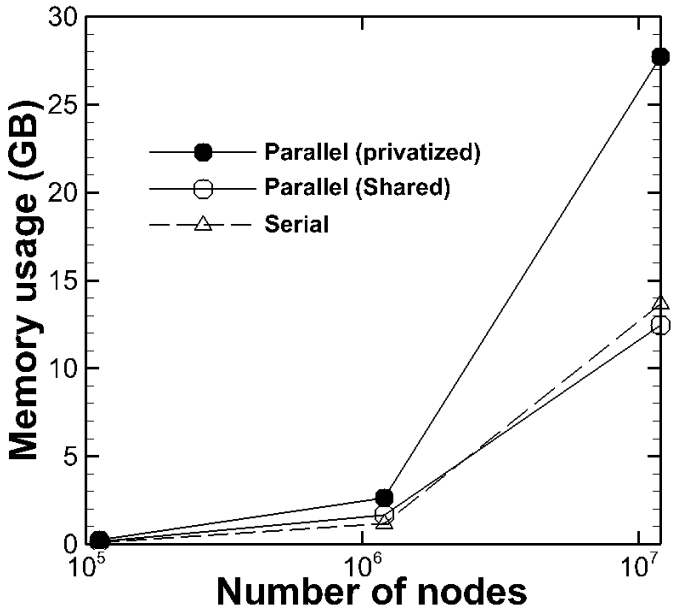

Figure 3.42 Memory use as a function of the number of mesh nodes for variably-saturated flow

3.8.2 Memory Usage

Before performing parallel simulations, it is important to determine if a computing machine can meet the system requirements for a simulation as well as provide reasonable computing times. Among the system requirements, the available memory in the system needs to meet the demand. To investigate the memory size necessary for a variably-saturated flow simulation, the requirements for three types of computations as a function of grid size are plotted in Figure 3.42. It can be seen that the memory use increases with the number of mesh nodes for variably-saturated flow. If the number of nodes is less than 10^5, the memory required for a simulation is less than 1.0 GB. Parallel computing with the privatization scheme needs 0.25 GB, which is almost twice as much as the serial case requires. For a simulation with 10^6 nodes, the memory size needed for the simulation is 2.6 GB for the parallel simulation with privatization, 1.6 GB for the parallel computing without privatization and 1.1 GB for the serial computation. If the simulations with and without privatization are compared for the case of 10^7 nodes, the privatization approach needs 27.7 GB, which is about 2.2 times more than the 12.5 GB of memory used for the shared scheme.

3.8.3 Parallel Efficiency

Figure 3.43 shows the distribution of maximum speedup values with 8 threads for matrix assembly and for the matrix solver, obtained from the simulations performed on GPC using PHGS optimized for speed. The maximum speedup for the solver ranges from 1.6 (variably-saturated flow) to 4.8 (solute transport) and that for assembly ranges from 3.5 (solute transport) to 7.1 (integrated surface-subsurface flow with unstructured meshes). The gray arrows in Figure 3.43 indicate a pair of the parallel speedup results obtained using the shared and privatized matrix schemes for the same simulation case. The results consistently indicate that the privatized scheme improves the maximum speedup for the matrix solver. Specifically, the privatization scheme improves the computing speed for the solver by about 47 % for saturated flow, 36 % for solute transport, 74 % variably-saturated flow, and 48 % for the integrated surface-subsurface flow on average.

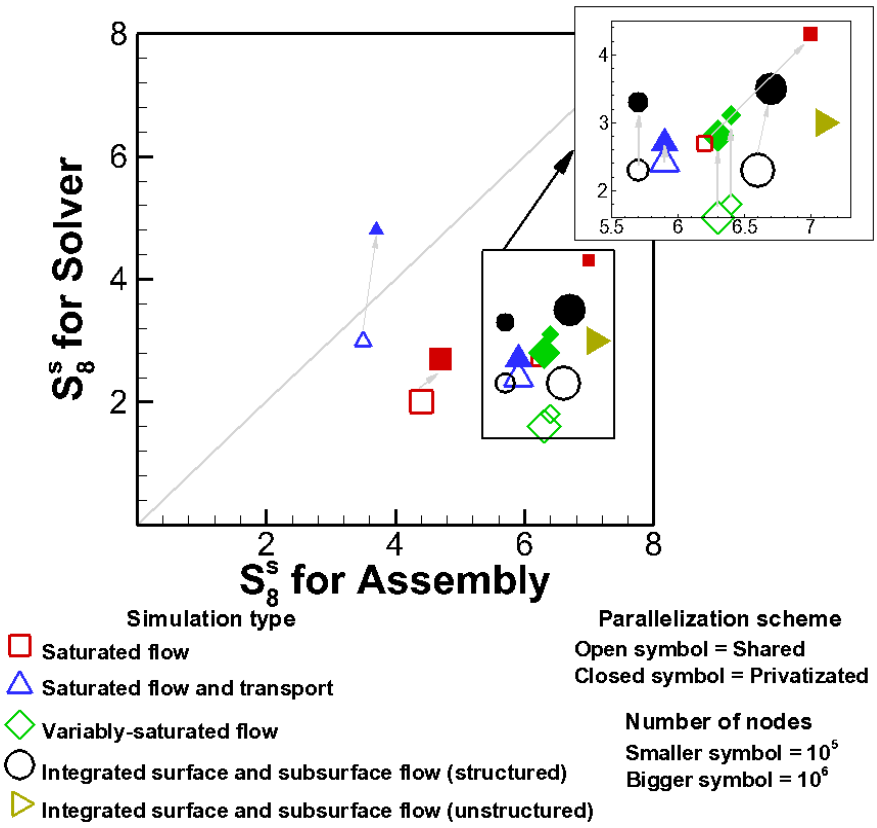

Figure 3.43 Distribution of maximum speedups, S_8^S, for matrix assembly and the matrix solver: the simulations were performed on GPC with PHGS being optimized for speed.

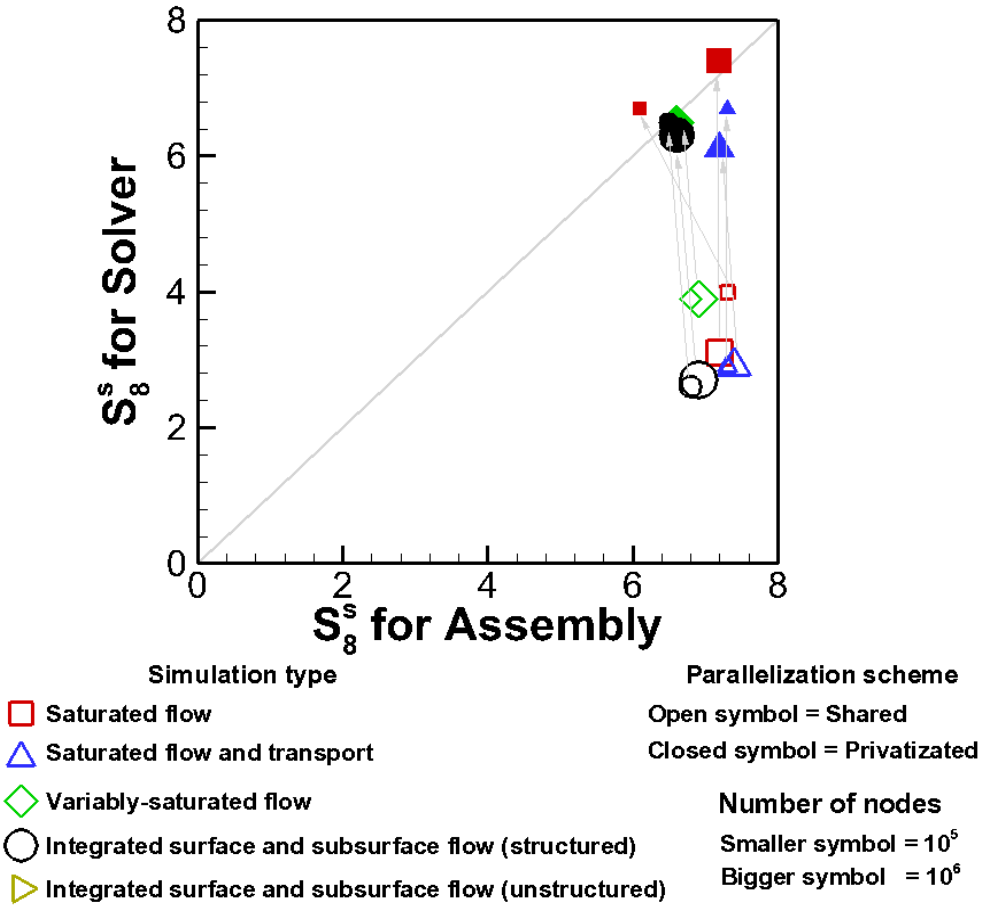

Figure 3.44 Distribution of maximum speedups, S_8^S, for matrix assembly and solver: the simulations were performed on GPC without code optimization for speed.

Figure 3.44 shows the maximum speedup using eight threads on GPC for matrix assembly and for the matrix solver for the various test simulations using the non-optimized version of PHGS. The maximum speedup for assembly ranges from 6.1 (saturated flow) to 7.4 (solute transport), while that for the solver ranges from 2.6 (integrated surface and subsurface flow with structured meshes) to 7.4 (solute transport). In Figure 3.44, it is clear that the speedup for the solver can be significantly improved by privatizing the sub-matrices. The maximum improvement for the matrix solver obtained by using the privatization is about 150 % for the integrated surface-subsurface flow simulation consisting of 10^5 mesh nodes. Specifically, the privatization scheme improves the computing speed for the matrix solver by about 103 % for saturated flow, 121 % for solute transport, 63 % for variably-saturated flow, and 142 % for the integrated surface-subsurface flow on average.

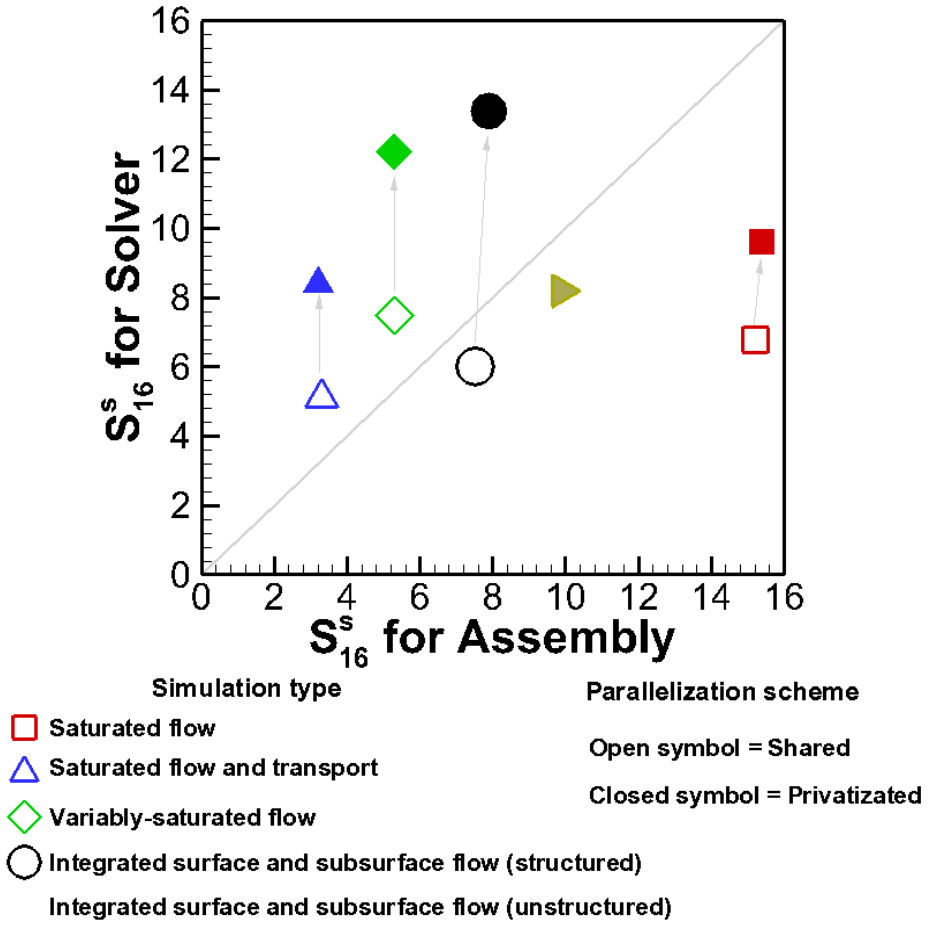

Figure 3.45 Distribution of maximum speedups, S_{16}^{S}, for matrix assembly and solver: the simulations were performed on TCS with a code optimization for speed.

Figure 3.45 shows maximum speedups, S_{16}^{S}, obtained for the various simulations consisting of 10^7 nodes on TCS. The maximum speedup for assembly ranges from 3.2 (solute transport) to 15.4 (saturated flow), while for the matrix solver, it is within a relatively narrow range from 5.1 (solute transport) to 13.4 (integrated surface-subsurface flow). The gray arrows in Figure 3.45 indicate the change between the shared and privatized matrix schemes. It is again consistent that the privatization improves the speedup for the solver by about 41 % for saturated flow, 65 % for solute transport, 63 % for variably-saturated flow and 123 % for integrated surface-subsurface flow.

Chapter 4
Parallel Simulations with High-Resolution Irregular Meshes

4.1 Introduction

Mesh resolution can significantly affect the simulation results [58].Simulations performed with a low-resolution grid may not accurately capture physical processes in localized regions of interest and may thus produce misleading results. Specifically, integrated surface-subsurface water flow simulations involve many interacting processes of various degrees of complexity caused by irregular terrain topography, atmospheric interactions, land surface and subsurface heterogeneity, fast stream flows, interactions between groundwater and surface water, and evapotranspiration.

Mesh refinement is a common approach to simulate detailed processes within a certain area. This section introduces a mesh refinement technique that facilitates process resolution when performing integrated surface-subsurface simulations consisting of triangular or prism meshes. However, simulations performed at high-resolution may cost too much time as the complexity of the physical processes increase. It is also important to utilize a reasonable initial condition to guarantee the convergence of the model and to save computing time. Two strategies to reduce computational costs are 1) to spin-up the high-resolution model with a reasonable initial condition and 2) to use parallel computation. A reasonable initial condition can be obtained from coarse mesh simulation results. A linear interpolation method is then used when refining the coarse mesh and it is also utilized for mapping the initial condition onto the refined meshes. Parallel computations on the fine grid combined with the mapped spin-up results from a coarse grid can significantly reduce computing costs. The parallel efficiency with the approach is evaluated in this chapter for one example involving a small-scale rainfall-runoff scenario and another centred on the 3D Canada-scale case study.

4.2 Mesh Refinement

The mesh refinement technique applied in this study generates a new node at the center of each line joining two nodes in a given triangle in irregular triangular mesh system. With this approach, three new nodes and four new triangular elements are generated for a single original triangular element. After the refinement process, the number of nodes is approximately quadrupled and the number of elements is quadrupled compared to the original coarse mesh.

Figure 4.1 illustrates the process of mesh refinement. For the original triangular element consisting of three nodes, three additional nodes are generated on the lines between two nodes. The numbering for the newly generated nodes starts from $nn2d_i$, the total number of nodes on layer i, to preserve the initial and boundary conditions originally assigned. The numbering method prevents conflicts caused by the addition of the newly generated nodes because the original nodes can easily be traced back in terms of the properties assigned to the nodes.

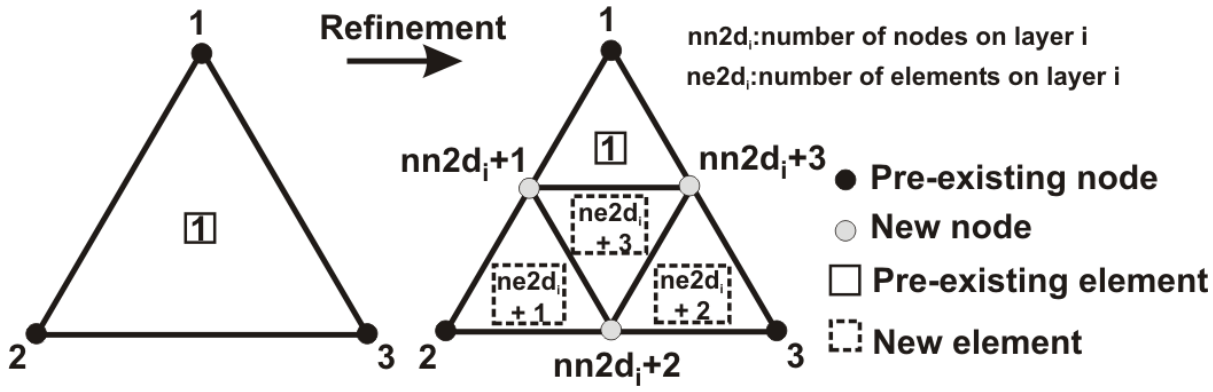

Figure 4.1 Irregular mesh refinement process

The numbering scheme for the new nodes is also counter-clockwise based on the first-generated node ($nn2d_i + 1$) on an element. Based on the generated nodes, the numbering of the elements is then performed. An element is divided into four fine elements when the refinement is performed. For numbering the newly-generated elements, the element is assigned its original element number (i.e., 1) if there is a node whose number is the smallest among the original node numbers. Numbering of the remainder of the elements starts from $ne2d_i + 1$, in which $ne2d_i$is the number of elements in layer i.

In terms of assigning nodal and elemental properties such as initial and boundary conditions to a refined domain, it is important to take into account the reconstructed nodes and elements. If the nodal and elemental properties are the same as those originally assigned to the coarse domain, the following example shows how to assign the properties.

4.2.1 Test Simulation with Mesh Refinement

A test simulation is performed based on a rainfall-runoff case study provided by [59]. A constant rainfall rate is applied to the surface domain at a rate of 2 cm/h until the system reaches a steady state. The parameters used in the simulation are listed in Table 4.1. The 3-D simulation domain consists of 15 layers with 1372 nodes and 2651 elements in each layer. For boundary conditions assigned to the surface flow domain, critical depth boundary condition is assigned along the stream channel outlet and the remaining boundaries over the subsurface domain are assumed to be impermeable.

Table 4.1 Parameters for the rainfall-runoff example

parameter	value
Surface	
X friction factor [-]	0.3
Y friction factor [-]	0.3
Rill storage height [-]	0.002
Obstruction storage height [m]	0.0
Coupling length [m]	0.1
Subsurface	
Porosity [-]	0.34
Specific storage [-]	1.2×10^{-7}
Saturated hydraulic conductivity [m/s]	1.2×10^{-5}
van Genuchten parameter α [1/m]	1.9
van Genuchten parameter β [m]	6.0
Initial water table depth (below ground surface) [m]	0.22

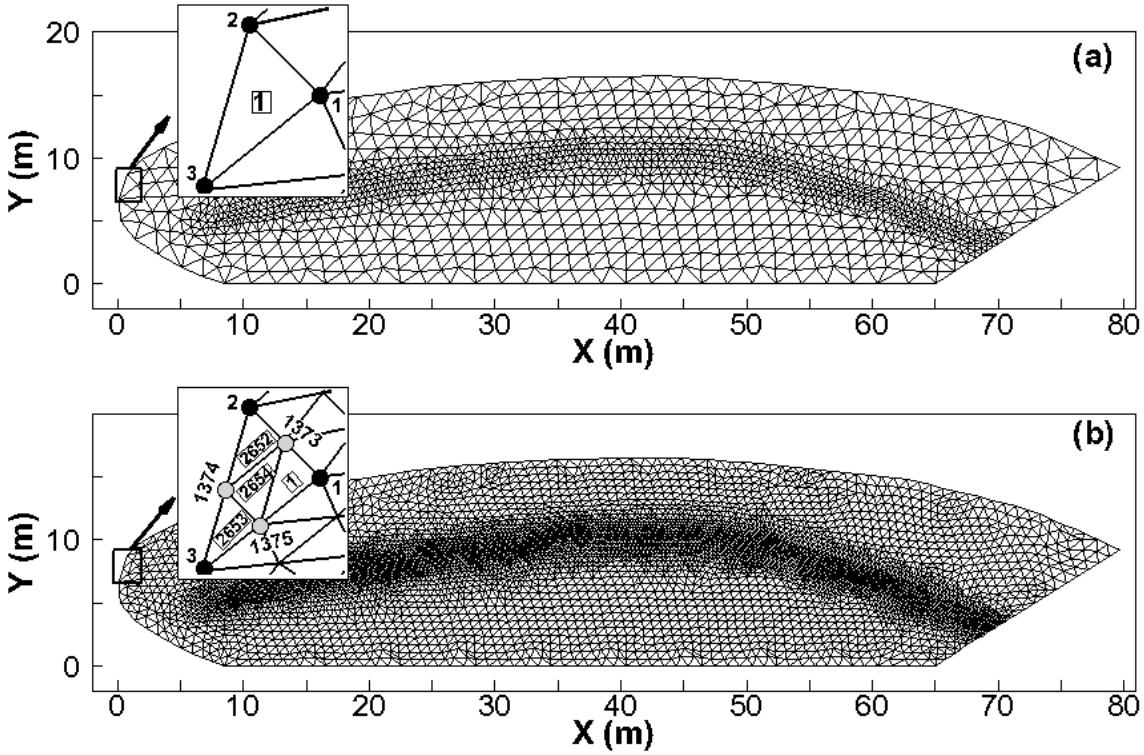

Figure 4.2 Comparison between coarse and fine meshes after mesh refinement for the rainfall-runoff example: a) coarse mesh; b) refined mesh

Figure 4.2 compares the original mesh to the refined one. All the elements in the coarse mesh are evenly divided into four smaller elements. Specifically, the number of nodes ($nn2d_1$) and the number of elements ($ne2d_1$) for the coarse mesh shown in Figure 4.2a become to 5394 nodes and 10604 elements (Figure 4.2b) after the mesh is refined. In terms of numbering the nodes and elements, because $nn2d_i$ for the coarse mesh is 1372, the new node numbers start from 1373 (Figure 4.2b). Similarly, the newly-generated elements are numbered from 2652 because $ne2d_1$equals 2651 (Figure 4.3b).

Once mesh is refined, it is necessary to assign the nodal and elemental boundary conditions such as specified heads, critical depths and fluxes based on the conditions assigned to the original mesh. The method for assigning boundary conditions is simply based on an interpolation approach applied to the mesh refinement. Elemental material properties for the new elements are based on those of the coarse elements of which they fall within.

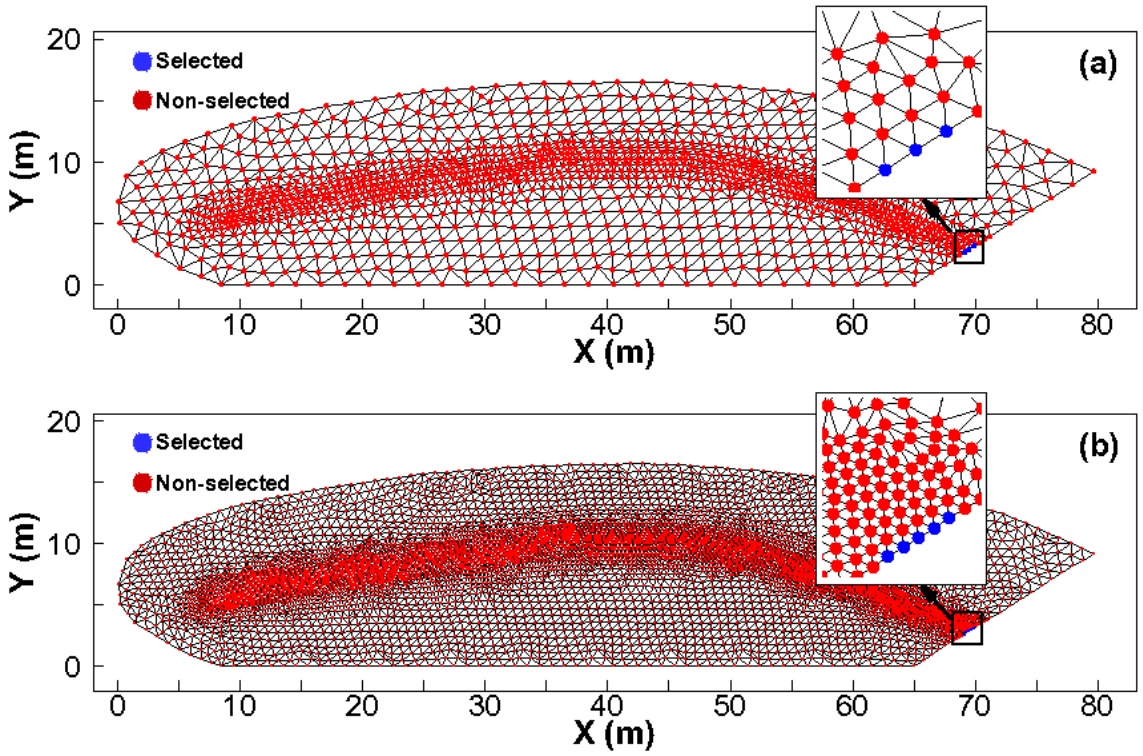

Figure 4.3 Boundary condition assignment for coarse and fine meshes for rainfall-runoff example:: a) coarse mesh; b) refined mesh

As an example for assigning nodal properties, if two adjacent nodes are assigned to have specified-head boundary conditions in the original mesh, the new node generated between the two original nodes during the mesh refinement will be chosen to be the specified head node. The selection of critical-depth boundary conditions along the stream outlet between the coarse and refined meshes is depicted in Figure 4.3. As deficted in Figure 4.3a, three nodes are originally assigned as the critical-depth boundary nodes. After mesh refinement, two additional nodes are assigned as critical-depth boundary nodes (Figure 4.3b). Thus, in total, five nodes are selected as critical-depth boundary conditions in the refined domain (Figure 4.3b).

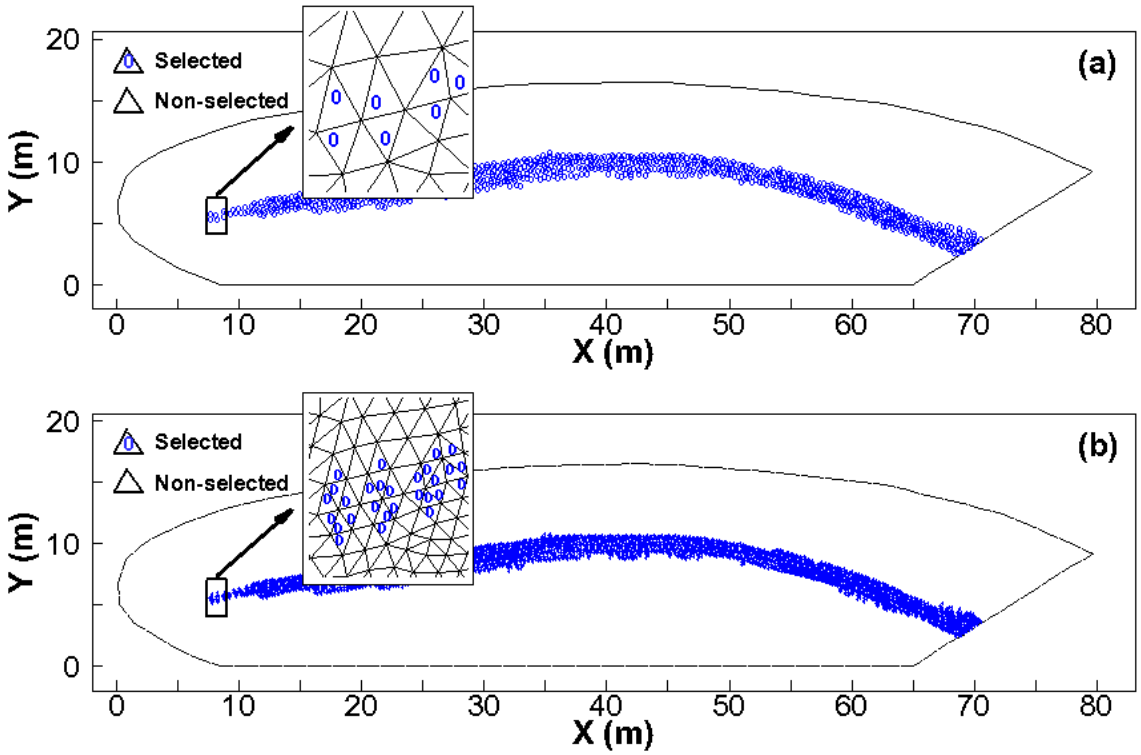

Figure 4.4 Comparison between coarse and fine meshes illustrating stream elements in rainfall-runoff example: a) coarse mesh; b) refined mesh

For the case of assigning elemental properties, the method is different from the one applied to the nodal conditions. With regard to the elements, the original mesh structure is preserved even though mesh refinement is performed, because each original element is evenly divided into four fine elements. The four fine elements are assigned the same property as that originally assigned to the coarse element. Figure 4.4 illustrates the distribution of stream channel elements for the coarse (Figure 4.4a) and fine meshes (Figure 4.4b). The total number of stream elements assigned is 868 for the coarse mesh and increases to 3472 after the mesh is refined.

4.2.2 A Spin-up Strategy for Simulations with the Refined Mesh

It is well known that for transient numerical simulations, it is important to take a reasonable initial condition. This is particularly important for integrated surface-subsurface simulations because initial values of surface water depths, stream flow rates, subsurface hydraulic head and soil saturations are needed, and factors such as irregular topography and heterogeneity can make the selection of a consistent initial condition extremely difficult. If a poor initial condition is selected, the excessive computing time is needed to spin-up the model to an equilibrium initial condition from which a transient simulation can proceed. One of the strategies to select a reasonable initial condition is to spin-up the model with a coarse grid to an equilibrium condition and to use the result as an initial condition on a fine-grid. To take advantage of using the simulation results obtained from the coarse mesh, the elemental interpolation method is applied to assign initial conditions to the newly-generated nodes. Specifically, the elemental interpolation function is applied to assign values to the new nodes. This approach is the same as using the elemental interpolation functions for a 2-D triangular element as follows:

$$\hat{u}(x, y) = \omega_1 u_1 + \omega_2 u_2 + \omega_3 u_3 \tag{4.1}$$

where $\hat{u}$ is an interpolated value at a point of (x, y); u_i is a known value at the elemental node i, at coordinate location (x_i, y_i); and ω_1, ω_2, and $\omega_3$are element basis (interpolation) functions. If a point to be interpolated is located at the center of two nodes, the values for the basis function associated with two nodes is 0.5, and the other is zero. Thus, the value at the center of two nodes is the average between two values at two nodes. For example, an interpolated value at the center of nodes 1 and 2 $\left(\frac{x_1+x_1}{2}, \frac{y_1+y_2}{2}\right)$ can be computed by

$$\hat{u}\left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right) = 0.5u_1 + 0.5u_2 \tag{4.2}$$

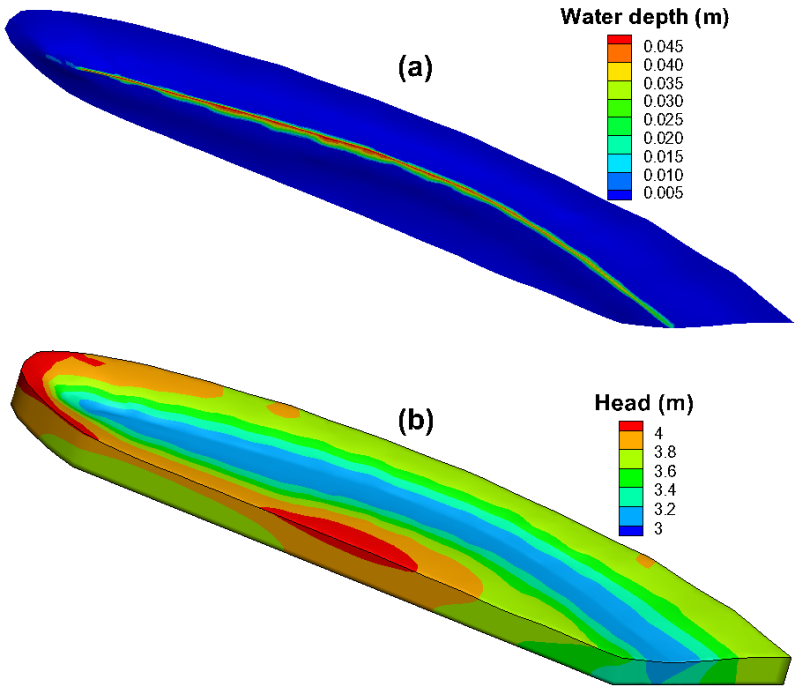

Figure 4.5. Simulation results with refined mesh for rainfall-runoff example: a) surface water depths; b) subsurface hydraulic heads

4.2.3 Simulation Results and Computing Efficiency for the Refined Mesh

The simulation results at steady state obtained with the refined mesh are shown in Figure 4.5. Note that in this case, the initial condition was established by spinning up the model using the fine mesh. On the land surface, the maximum water depth is 4.6×10^{-2} m, which occurs at the centre of the stream, while the minimum water depth equal to 2.2×10^{-3} m occurs at the edges of the stream. In the subsurface domain, the hydraulic heads along upslope regions are higher than those in the centre of the stream because of the elevation difference.

To verify the robustness of the fine mesh simulation using an interpolated initial condition obtained from the coarse mesh results, a simulation was performed with the fine mesh but with the initial conditions taken from the spinning up with the coarse mesh and interpolated onto the fine mesh. The maximum difference in water depths between the two fine-mesh simulations with equilibrium initial conditions taken from the coarse and fine meshes are less than 10^{-6} m, which is negligible. The maximum head difference in the subsurface is less than 10^{-6} m, which is again considered to be negligible

Table 4.2 Comparison of simulation performance results using fine mesh and initial conditions based on 1) poor guess on fine mesh and 2) converged initial condition interpolated onto fine mesh from coarse mesh spin-up.

		Simulation 1[1)]	Simulation 2[2)]
Time step		457	41
Iteration	**Newton-Raphson**	1748	116
	Iterative solver	72954	5143
Total computing time (sec)		3340	239

[1)]: fine-grid simulation with poor initial condition
[2)]: fine-grid simulation with converged initial condition taken from coarse grid

The fine-grid simulation results shown in Figure 4.5 were obtained using a trivial (poor) guess for the initial conditions and, as such, the model had to be spun up to achieve an equilibrium initial condition for the subsequent transient simulation. The computational effort for this case, Simulation 1, is provided in Table 4.2. Simulation 2 results also shown in Table 4.2 were also obtained with a fine grid, but the spin up process on the fine grid was avoided because the converged initial condition was taken from the coarse mesh and interpolated onto the fine mesh. The performance results for the two approaches are compared in Table 4.2. Because integrated surface-subsurface models solve non-linear partial differential equations, the Newton-Raphson method was applied to iteratively solve the nonlinear discrete equations. Thus, the measures used for the comparison are the total number of time steps taken to reach steady state, the number of Newton-Raphson and matrix solver iterations, and the total computing time.

The total number of time steps taken for simulation 1 is 457, while that for simulation 2 is 41, which is ten times. The number of Newton-Raphson and solver iterations for simulation 1 are 1748 and 72954, respectively. For simulation 2, the number of iterations is 116 Newton-Raphson loops and 5143 solver iterations. Thus, simulation 1 takes almost ten times as many solver iterations as does simulation 2. The total computing time for simulation 1 is 3340 sec, which about 14 times more than that for simulation 2. Thus, based on the comparison between the two simulations, using equilibrium initial conditions based on a simulation employing a coarse mesh can play a pivotal role in increasing computational efficiency.

4.3 Real-World Simulations with a High Resolution Mesh

The results from the rainfall-runoff test problem demonstrate that a considerable saving in computing effort can be achieved by using an initial (spin-up) condition derived from a coarse mesh and then interpolated onto a fine mesh. This saving becomes even more substantial for large-scale simulations performed using a fine mesh. In this section, we return to the 3D Canada-scale surface-subsurface example described in section 3.7, but will repeat the computation using a fine mesh and using the equilibrium initial condition taken from the coarse mesh and mapped onto the fine mesh.

The surface domain for the fine grid consists of 230063 nodes with 452424 elements, while the subsurface domain consists of 3681008 nodes with 6786366 elements. A combined total of 3911071 nodes and 7238784 elements are in the the full simulation domain. The boundary conditions and physical properties are identical to those used in the coarse grid example provided in section 3.7.

4.3.1 Simulation Results

The fine-grid simulation results for the surface water depths and water exchange fluxes between the surface and subsurface flow regimes are compared to those obtained with the coarse mesh in Figure 4.6. The distribution of surface water depths for the refined and coarse mesh are similar because land surface elevations are the dominant factor affecting overland flow rates and water depths [60]. The fine-grid results, however, provide much improved resolution of the results. Nevertheless, several differences such as the locations of water bodies and surface water depths can be identified. Specifically, Lakes Superior, Huron, and Michigan shown by the magnified views illustrate the difference in water depth (Figure 4.6). Based on the topographic map (a white dashed circle) on the right, Lake Michigan and Lake Huron are connected and considered as a continuous surface water body. Thus, the refined mesh provides a stronger connection between the two lakes, even if the general topography is similar for both models.

The distribution of water exchange fluxes for the coarse and refined grids is shown in Figure 4.7. The positive exchange flux indicates that the water fluxes are from the subsurface domain to the surface domain. Thus, positive exchange fluxes can be considered as groundwater discharge areas. Overall, the distributions of exchange fluxes for the coarse and refined cases are similar with positive

values occurring along streams and lakes. However, focusing on Lake Superior, Lake Michigan and Lake Huron, the recharge-discharge divides near the lakes are slightly different between the meshes. The improvement in the resolution of the recharge-discharge patterns obtained with the fine mesh is clearly evident.

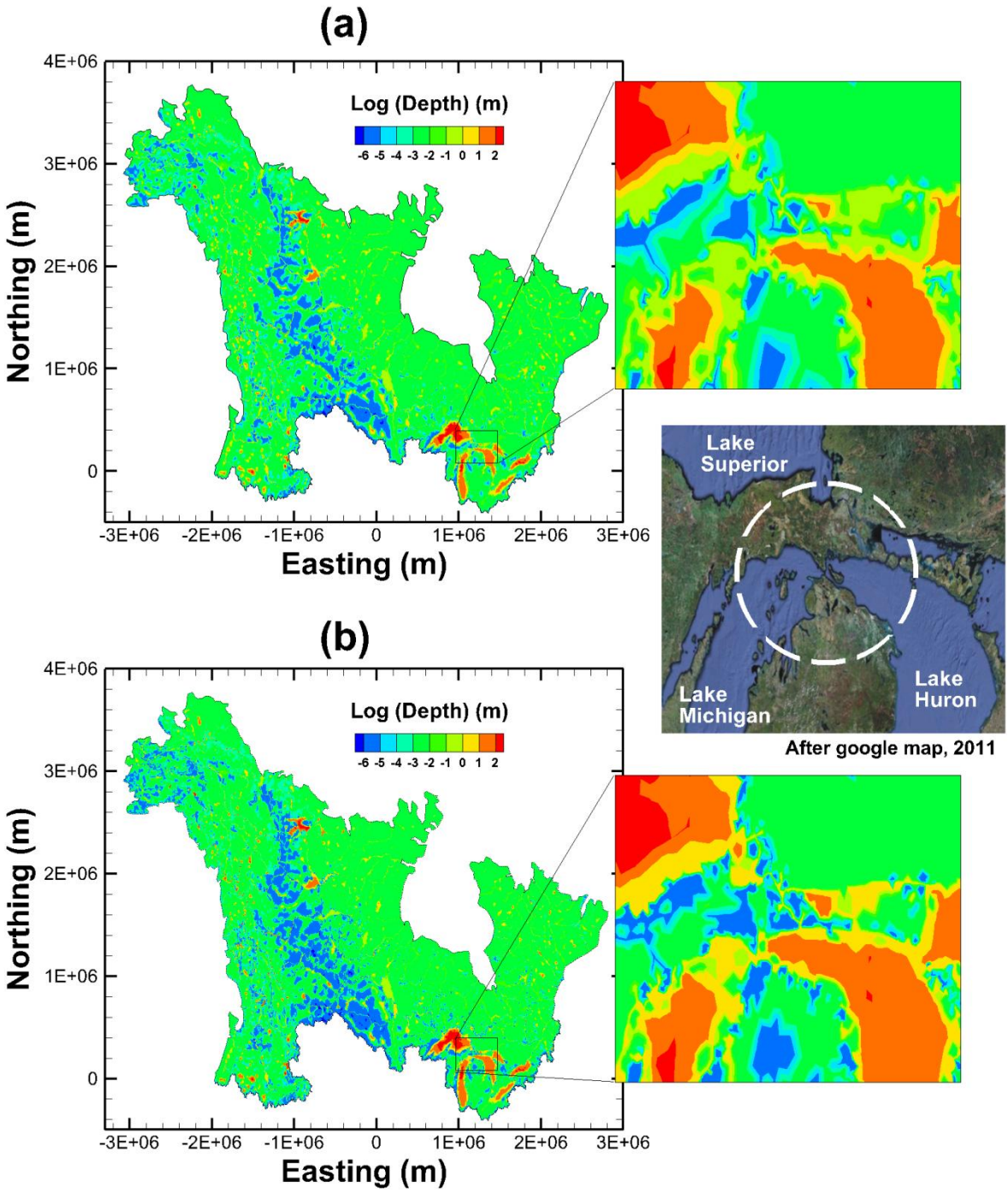

Figure 4.6 Comparison of simulation results for surface water depths between (a) coarse and (b) refined meshes. The box at the right shows a magnified view.

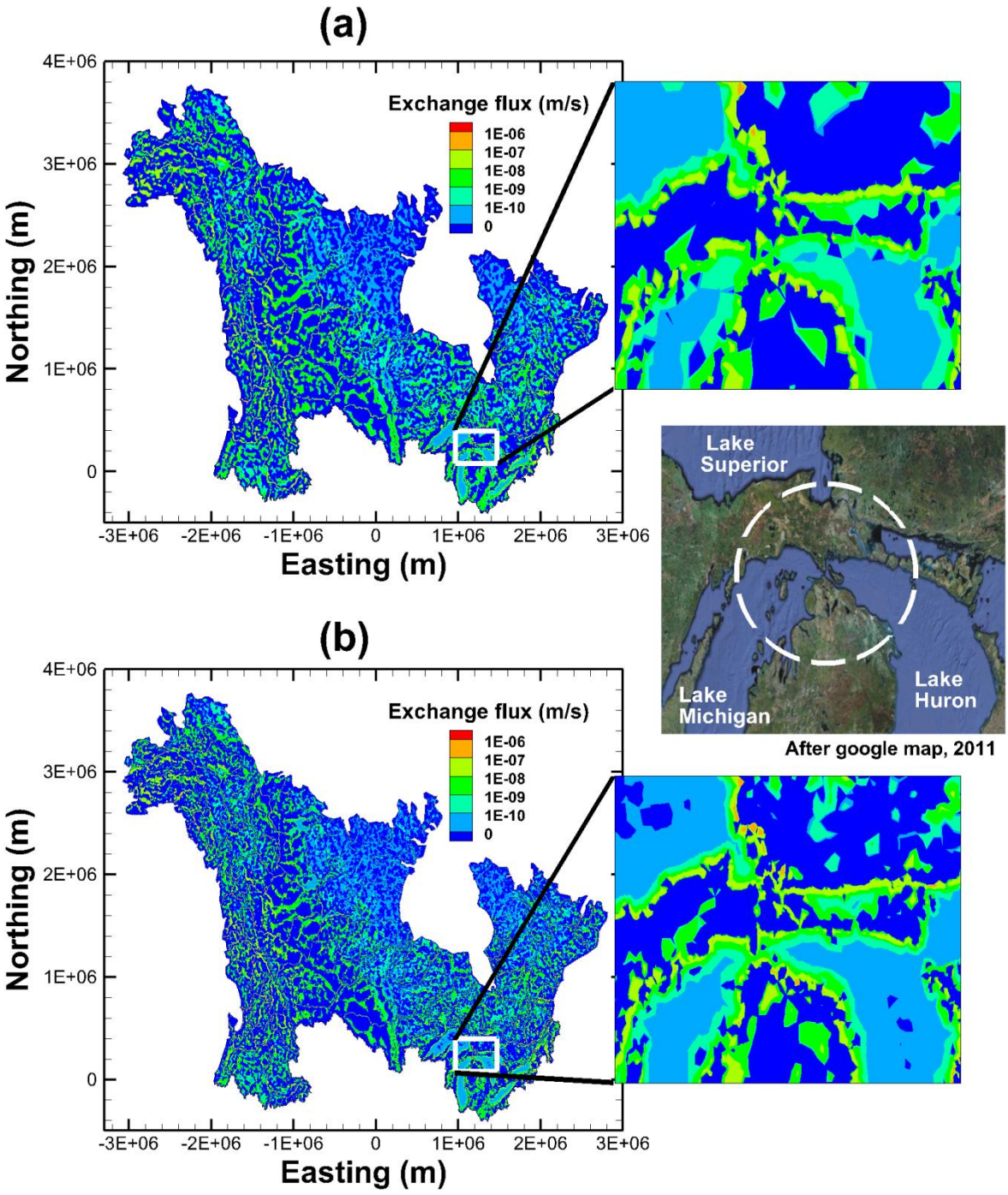

Figure 4.7 Comparison of simulation results for water exchange fluxes between (a) coarse and (b) refined domains. The box at the bottom right shows a magnified view.

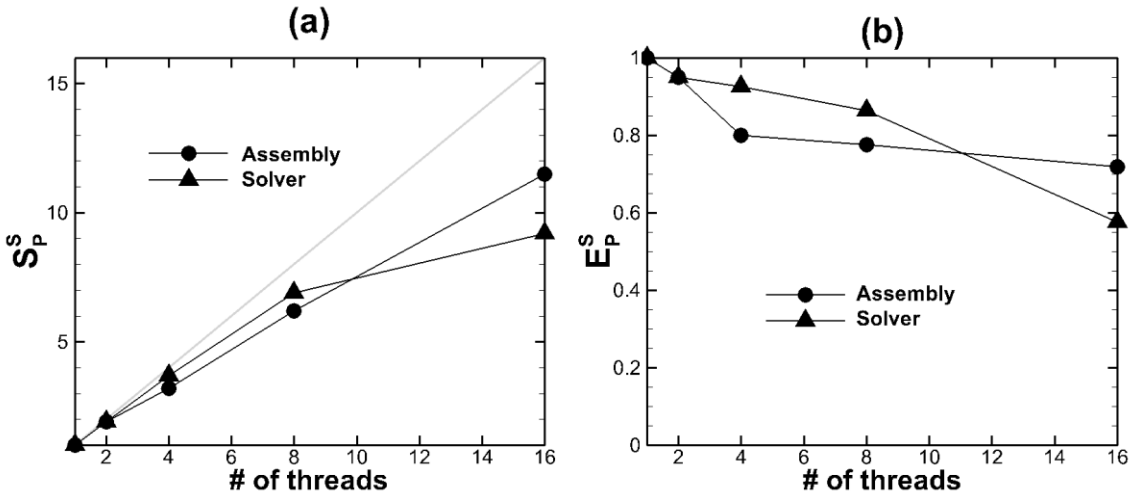

Figure 4.8 Results of speedup and parallel efficiency for refined large-scale simulation performed by TCS: a) parallel speedup, b) parallel efficiency

4.3.2 Parallel Efficiency for the Simulation with Refined Mesh

The parallel speedup, S_P^S, and parallel efficiency, E_P^S, for the refined large-scale simulation performed on TCS are shown in Figure 4.8. The calculation of S_P^S for assembly is based on the time required for a single matrix assembly, while S_P^S for the matrix solution is based on the computing time per single solver iteration. S_P^S for assembly and for the solver increase with the number of threads (Figure 4.8a). The improved rate for both tasks is similar to each other, and the speedup for assembly is relatively high. The maximum speedup for the solver is 11.5 ($E_{16}^S = 0.72$).

The parallel efficiency, E_P^S, for both tasks decrease with the number of threads from 0.96 (=E_2^S) to 0.72 (=E_{16}^S) for the solver and from 0.97 (=E_2^S) to 0.58 (=E_{16}^S) for assembly. A comparison of the total computing time between the coarse mesh simulation and that obtained with the fine mesh, with a poor initial guess cannot be made because this latter case could not be simulated due to excessive computing time. Clearly, this is an indication that performing fine-grid simulations for large-scale surface-subsurface problems demand an accurate initial equilibrium condition. A workable way to obtain the initial spin-up condition is therefore to perform this step first on a coarse grid and then map the results onto the fine mesh.

Chapter 5
Summary and Conclusions

Changing climate will have significant impacts on future water resource and thus the analysis of the impacts may need to consider a wide spectrum of factors influencing surface-subsurface water resource such as climate, agriculture, land use, ecosystem, and forest managements. Because modern fully-integrated surface-subsurface hydrologic models attempt to capture many complex interacting processes, high performance computing (HPC) is essential. One of the most straightforward ways to adapt HPC for hydrological applications is to utilize pre-developed HPC packages or libraries such as Aztec, DBuilder, hypre, KINSOL and PETSc. However, HPC library packages are not suitable for general hydrological models because they do not provide a library for the Jacobian matrix assembly, which consumes most of the execution time for hydrologic simulations. The main objective of this research was to apply parallel frameworks to enhance the performance of HydroGeoSphere (HGS) and to develop an efficient and flexible parallel interface using OpenMP, which can be performed on most personal computers.

The computational bottle neck within various hydrologic simulations was analyzed to develop the targeted parallelization techniques for improving the overall computing efficiency. The computing tasks consisting of matrix assembly, matrix solution and preconditioning, are main parts to be parallelized since they require more than 90 % of the total computing time for saturated flow, solute transport, variably-saturated flow and integrated surface-subsurface flow simulations. Matrix assembly was parallelized using a coarse-grained scheme because the Jacobian matrix can be constructed independently. The average parallel speedup of the matrix assembly process is 6.3 (=S_8^Son GPC) and 15.3 (= S_{16}^S on TCS) for saturated flow, 6.0 (=S_8^Son GPC) and 3.3 (= S_{16}^S on TCS) for solute transport, 6.5 (=S_8^Son GPC) and 5.3 (= S_{16}^S on TCS) for variably-saturated flow, and 6.4 (=S_8^Son GPC) and 7.7 (= S_{16}^S on TCS) for integrated surface-subsurface flow. The parallelization of the iterative solver, BiCGSTAB, was achieved using domain partitioning and privatization schemes. It is clear that total speedup for the test simulations increases with the number of threads used. The average parallel speedup of the solver obtained with privatization is 4.1 (=S_8^Son GPC) and 8.2 (= S_{16}^S on TCS) for saturated flow, 3.9 (=S_8^Son GPC) and 6.8 (= S_{16}^S on TCS) for solute transport, 3.7 (=S_8^Son GPC) and 9.9 (= S_{16}^S on TCS) for variably-saturated flow, 3.7 (=S_8^Son GPC) and 9.7 (= S_{16}^S on TCS) for the integrated surface-subsurface flow. The privatization scheme improves the average computing

speed for the solver by about 75 % for saturated flow, 78 % for solute transport, 68 % variably-saturated flow, and 95 % for the integrated surface-subsurface flow.

A large-scale real-world simulation involving tightly-coupled, highly-nonlinear feedback conditions was conducted to evaluate the applicability, accuracy, robustness and scalability of Parallel HydroGeoSphere. The average parallel speedup was 7.1 (=S_8^Son GPC) and 9.9 (= S_{16}^S on TCS) for matrix assembly and 3.0 (=S_8^Son GPC) and 8.2 (= S_{16}^S on TCS) for the solver with privatization. This complex simulation demonstrated the importance of node reordering, which is a process involved in domain partitioning. With node reordering, super-linear scalarable parallel speedup was obtained by comparing serial jobs performed with the natural node ordering. Specifically, the speedup for the solver with privatization was 8.6 (=S_8^Son GPC) and 23.5 (= S_{16}^S on TCS). Based on the various simulations performed for evaluating parallel efficiency, the parallelized BiCGSTAB using privatization scheme is stable and efficient.

Mesh refinement is a common approach to capture detailed hydrologic processes within a certain target area. However, simulations performed on at high-resolution meshes may require excessive computing time. Thus, it is important to initiate a simulation with a reasonable initial condition to guarantee the convergence of the model and to save computing time. A straightforward method was applied to refine meshes, and to select a reasonable initial condition based on a coarse grid and then mapped onto the fine mesh. The mesh refinement and spin-up approaches were tested on small- and large-scale problems. The spin-up method for the fine mesh simulations accelerates the convergence of the simulations and can play a pivotal role in increasing computational efficiency.

Using the parallel schemes developed in this work, three key achievements can be summarized: First, parallelization of physically-based hydrologic simulations such as HydroGeoSphere using OpenMP for shared memory machines allows the same code to be executed on multiple platforms such as multicore desktops, laptops as well as supercomputing machines. Second, a general and flexible iterative sparse-matrix solver was parallelized and thus parallel solver can be implemented in a wide range of numerical methods employing either structured or unstructured mesh. Moreover, a higher parallel efficiency of the matrix solver was achieved using the privatization scheme, which is developed in this work. Lastly, compared to other parallelized hydrologic models, which use parallel library packages, pHGS is flexible for model modifications and upgrades especially for constructing the coefficient and Jacobian matrixes.

Bibliography

[1] Tuminaro RS, M Heroux, SA Hutchinson, JN Shadid. Official Aztec User's Guide, version 2.1. Massively Parallel Computing Research Laboratory, Sandia National Laboratories, Albuquerque, NM, 1999.

[2] Hunter RM, JRC Cheng. DBuilder: A parallel data management toolkit for scientific applications. Las Vegas, NV, (2005) pp. 825-31.

[3] Falgout RD, UM Yang. hypre: A library of high-performance preconditioners. Center for applied scientific computing Lawrence Livermore National Laboratory, Livermore, CA, 2002.

[4] Collier AM, AC Hindmarshand, R Serban, CS Woodward. User Documentation for KINSOL v2.6.0. Lawrence Livermore National Laboratory, Livermore, CA, 2009.

[5] Balay S, WD Gropp, LC McInnes, BF Smith. PETSc users manual. Argonne National Laboratory, Argonne, IL 2002.

[6] Mills RT, C Lu, PC Lichtner, GE Hammond. Simulating subsurface flow and transport on ultrascale computers using PFLOTRAN. Journal of Physics: Conference Series. 78 (2007).

[7] Zhang K, YS Wu, K Pruess. User's Guide for TOUGH2-MP - A Massively Parallel Version of the TOUGH2 Code. Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 2008.

[8] DHI-WASY GmbH. FEFLOW finite element subsurface flow and transport simulation system—User's manual/reference manual/white papers. Recent release 6.0. Tech. rep. DHI-WASY GmbH, Berlin 2010.

[9] Wang W, G Kosakowski, O Kolditz. A parallel finite element scheme for thermo-hydro-mechanical (THM) coupled problems in porous media. Comput Geosci-Uk. 35 (2009) pp. 1631-41.

[10] Maxwell RM, SJ Kollet, SG Smith, CS Woodward, RD Falgout, IM Ferguson, et al. ParFlow User's Manual. International Ground Water Modeling Center2010.

[11] Cheng J-RC, RM Hunter, H-P Cheng, H-C Lin, DR Richards. Parallelization of the WASH123D Code---Phase II: Coupled Two-Dimensional Overland and Three-Dimensional Subsurface Flows. in: R Walton, (Ed.). 40792 ed. ASCE, Anchorage, Alaska, USA, (2005) pp. 279.

[12] van der Vorst HA. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. Siam J Sci Stat Comp. 13 (1992) pp. 631-44.

[13] Pruess K, C Oldenburg, G Moridis. TOUGH2 User''s Guide, V2.0. Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1999.

[14] Maxwell RM, NL Miller. Development of a Coupled Land Surface and Groundwater Model. Journal of Hydrometeorology. 6 (2005) pp. 233-47.

[15] Maxwell RM, JK Lundquist, JD Mirocha, SG Smith, CS Woodward, AFB Tompson. Development of a Coupled Groundwater–Atmosphere Model. Mon Weather Rev. 139 (2011) pp. 96-116.

[16] Ashby SF, RD Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. Nucl Sci Eng. 124 (1996) pp 145-59.
[17] Cheng J-RC, PE Plassmann. A Parallel Particle Tracking Framework for Applications in Scientific Computing. The Journal of Supercomputing. 28 (2004) pp. 149-64.
[18] Hammond G, P Lichtner, C Lu. Subsurface multiphase flow and multicomponent reactive transport modeling using high-performance computing. Journal of Physics: Conference Series. 78 (2007).
[19] Tuminaro RS, JN Shadid, SA Hutchinson. Parallel Sparse Matrix-Vector Multiply Software for Matrices with Data Locality. Sandia National Laboratories, Albuquerque, NM, 1997.
[20] Saad Y, MH Schultz. Gmres - a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear-Systems. Siam J Sci Stat Comp. 7 (1986) pp. 856-69.
[21] Freund RW. A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems. Siam J Sci Comput. 14 (1993) pp. 470-82.
[22] Saad Y. Krylov Subspace Methods on Supercomputers. Siam J Sci Stat Comp. 10 (1989) 1200-32.
[23] Wang H-C, K Hwang. Multicoloring of Grid-Structured PDE Solvers on Shared-Memory Multiprocessors. IEEE Trans Parallel Distrib Syst. 6 (1995) pp. 1195-205.
[24] Ma S. Comparisons of the parallel preconditioners for large nonsymmetric sparse linear systems on a parallel computer. International Journal of High Speed Computing. 12 (2004) pp. 55-68.
[25] Dongarra J, I Foster, G Fox, W Gropp, K Kennedy, L Torczon, et al. Sourcebook of Parallel Computing. Morgan Kaufmann, San Francisco, 2002.
[26] Wang Q, YC Guo, XW Shi. A Generalized Gps Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix. Prog Electromagn Res. 90 (2009) pp. 121-36.
[27] Boland D, GA Constantinides. Optimising Memory Bandwidth Use for Matrix-Vector Multiplication in Iterative Methods. Reconfigurable Computing: Architectures, Tools and Applications. 5992 (2010) pp. 169-81.
[28] Koohestani B, R Poli. A Genetic Programming Approach to the Matrix Bandwidth-Minimization Problem. Parallel Problem Solving from Nature-Ppsn Xi, Pt Ii. 6239 (2010) pp. 482-91.
[29] Wang H, F Liu, L Xia, BK Li, S Crozier. An Efficient BiCGstab Solved Impedance Method for Induced Field Evaluation with a Hyperthermia Applicator. Ieee Eng Med Bio. (2008) pp. 5636-9.
[30] Lee H-B, H-K Jung, S-Y Hahn, K Choi, H-J Kim. On the convergence rate improvement of ICCG solver on the FE mesh. Magnetics, IEEE Transactions on. 33 (1997) pp. 1760-3.
[31] Heath MT, P Raghavan. A Cartesian Parallel Nested Dissection Algorithm. Siam J Matrix Anal A. 16 (1995) pp. 235-53.
[32] Bader M, C Zenger. A fast solver for convection diffusion equations based on nested dissection with incomplete elimination. Euro-Par 2000 Parallel Processing, Proceedings. 1900 (2000) pp. 795-805.
[33] George A. Nested Dissection of a Regular Finite Element Mesh. Siam J Numer Anal. 10 (1973) pp. 345-63.

[34] Brainman I, S Toledo. Nested-dissection orderings for sparse LU with partial pivoting. Siam J Matrix Anal A. 23 (2002) pp. 998-1012.
[35] Gomperts R, M Frisch, J-P Panziera. Scalability of Gaussian 03 on SGI Altix: The Importance of Data Locality on CC-NUMA Architecture Evolving OpenMP in an Age of Extreme Parallelism. in: M Müller, B de Supinski, B Chapman, (Eds.). Springer Berlin / Heidelberg (2009) pp. 93-103.
[36] Grun P, N Dutt, A Nicolau. Access pattern based local memory customization for low power embedded systems. Design, Automation and Test in Europe, 2001 Conference and Exhibition 2001 Proceedings (2001) pp. 778-84.
[37] Tsai WF, CY Shen, HH Fu, CC Kou. Study of Parallel Computation for Ground-Water Solute Transport. J Hydrol Eng. 4 (1999) pp. 49-56.
[38] Wu YS, KN Zhang, C Ding, K Pruess, E Elmroth, GS Bodvarsson. An efficient parallel-computing method for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media. Advances in Water Resources. 25 (2002) pp. 243-61.
[39] Cooley RL. A Finite Difference Method for Unsteady Flow in Variably Saturated Porous Media: Application to a Single Pumping Well. Water Resour Res. 7 (1971) pp. 1607-25.
[40] Neuman SP. Saturated-unsaturated seepage by finite elements. ASCE Journal of the hydraulics division. (1973) pp. 2233-50.
[41] van Genuchten MT. A Closed-form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils. Soil Sci Soc Am J. 44 (1980) pp. 892-8.
[42] Brooks RJ, AT Corey. Hydraulic properties of porous media, Hydrology paper No 3, Colorado State university, Fort Collins, CO., 1964.
[43] Therrien R, RG McLaren, EA Sudicky, SM Panday. HydroGeoSphere. A three-dimensional numerical model describing fully-integrated subsurface and surface flow and solute transport. Waterloo, ON, 2010.
[44] Frind EO. Fundamentals of Groundwter Modeling, Waterloo, ON, 2003.
[45] Paniconi C, M Putti. A Comparison of Picard and Newton Iteration in the Numerical-Solution of Multidimensional Variably Saturated Flow Problems. Water Resour Res. 30 (1994) pp. 3357-74.
[46] Mehl S. Use of Picard and Newton iteration for solving nonlinear ground water flow equations. Ground Water. 44 (2006) pp. 583-94.
[47] Forsyth PA, RB Simpson. A 2-Phase, 2-Component Model for Natural-Convection in a Porous-Medium. Int J Numer Meth Fl. 12 (1991) pp. 655-82.
[48] Clift SS, EFD Azevedo, PA Forsyth, JR Knightly. WATSIT-1 and WATSIT-B, Waterloo Sparse, Iterative Matrix Solvers: User's Guide with Developer Notes for Version 2.0.0. University of Waterloo1997.
[49] Chin P, PA Forsyth. A comparison of GMRES and CGSTAB accelerations for incompressible Navier-Stokes problems. J Comput Appl Math. 46 (1993) pp. 415-26.
[50] Mahinthakumar G, F Saied. A Hybrid MPI-OpenMP Implementation of an Implicit Finite-Element Code on Parallel Architectures. International Journal of High Performance Computing Applications. 16 (2002) pp. 371-93.

[51] Scalasca toolset for scalable performance analysis of large-scale parallel applications. Supercomputing Centre
[52] Chapman B, G Jost, R van der Pas. Using OpenMP : portable shared memory parallel programming. The MIT Press, Cambridge, MA, USA, 1994.
[53] Jacques T, L Nicolas, C Vollaire. Electromagnetic scattering with the boundary integral method on mimd systems. Kluwer Int Ser Eng C. 515 (1999) pp. 215-30.
[54] Yang L, R Brent. The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. Fifth International Conference on Algorithms and Architectures for Parallel Processing, Proceedings. (2002) pp. 324-8.
[55] Brill SH, GF Pinder. Parallel implementation of the Bi-CGSTAB method with block red-black Gauss-Seidel preconditioner applied to the Hermite collocation discretization of partial differential equations. Parallel Computing. 28 (2002) pp. 399-414.
[56] Martorell X, M Gonzalez, A Duran, J Balart, R Ferrer, E Ayguade, et al. Techniques supporting threadprivate in OpenMP. Proceedings of the 20th international conference on Parallel and distributed processing. IEEE Computer Society, Rhodes Island, Greece. (2006) pp. 227-.
[57] Scinet. SciNet User Tutorial. SciNet/University of Toronto2010.
[58] Horritt MS, PD Bates, MJ Mattinson. Effects of mesh resolution and topographic representation in 2D finite volume models of shallow water fluvial flow. Journal of Hydrology. 329 (2006) pp. 306-14.
[59] Jones JP, EA Sudicky, RG McLaren. Application of a fully-integrated surface-subsurface flow model at the watershed-scale: A case study. Water Resour Res. 44 (2008) W03407.
[60] Loos M, H Elsenbeer. Topographic controls on overland flow generation in a forest – An ensemble tree approach. Journal of Hydrology, 2011.