Distributed memory parallel computing of three-dimensional variable-density groundwater flow and salt transport

Verkaik, J.; van Engelen, J.; Huizer, S.; Bierkens, Marc F.P.; Lin, H.X.; Oude Essink, G.H.P.

DOI
10.1016/j.advwatres.2021.103976

Publication date
2021

Document Version
Final published version

Published in
Advances in Water Resources

Advances in Water Resources 154 (2021) 103976

Contents lists available at ScienceDirect

Advances in Water Resources

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

Distributed memory parallel computing of three-dimensional variable-density groundwater flow and salt transport

J. Verkaik [a,b,*], J. van Engelen [a,b], S. Huizer [b,c], M.F.P. Bierkens [a,b], H.X. Lin [d,e], G.H.P. Oude Essink [a,b]

[a] Unit Subsurface and Groundwater Systems, Deltares, Utrecht, The Netherlands
[b] Department of Physical Geography, Utrecht University, Utrecht, The Netherlands
[c] Department Land & Watermanagement, Arcadis, Arnhem, The Netherlands
[d] Delft Institute of Applied Mathematics, Delft University of Technology, Delft, The Netherlands
[e] Institute of Environmental Sciences, Leiden University, Leiden, The Netherlands

ARTICLE INFO

Keywords:
Parallel computing
Distributed memory
Variable-density groundwater flow
Salt transport
Numerical modelling
SEAWAT

ABSTRACT

Fresh groundwater reserves, being of vital importance for more than a billion of people living in the coastal zone, are being threatened by saltwater intrusion due to anthropogenic activities and climate change. High resolution three-dimensional (3D), variable-density (VD), groundwater flow and salt transport (FT) numerical models are increasingly being used to support water managers and decision makers in their strategic planning and measures for dealing with the problem of fresh water shortages. However, these computer models typically require long runtimes and large memory usage, making them impractical to use without parallelization. Here, we parallelize SEAWAT, and show that with our parallelization 3D-VD-FT modeling is now feasible for a wide range of hydrogeologists, since a) speedups of more than two orders of magnitude can be obtained as illustrated in this paper, and b) large 3D-VD-FT models are feasible with memory requirements far exceeding single machine memory.

1. Introduction

Saltwater intrusion caused by anthropogenic activities and climate change threatens fresh groundwater reserves that are of vital importance for more than a billion of people living in the coastal zone (Neumann et al., 2015). Coastal, unconsolidated, groundwater systems are under pressure of salinization due to multiple threats that are related to climate change, population increase and associated increase of freshwater demand, and economic growth. Examples are land subsidence by excessive groundwater pumping (Minderhoud et al., 2017), surface sealing by urbanization (Renaud et al., 2013), climate-induced relative sea-level rise (Ferguson and Gleeson, 2012; Oude Essink et al., 2010), seawater-overwash by storm surges (Yang et al., 2013), and changing patterns for groundwater recharge, evaporation and groundwater seepage (Faneca Sanchez et al., 2012). To come up with strategies and measures to address these problems, water managers and policy makers require accurate, quantitative, future projections at the highest resolution possible. As a result, high-resolution, three-dimensional (3D), groundwater flow and salt transport models become more and more important instruments to support water coastal management and policy development (Harbo et al., 2011). Unfortunately, such models are generally very computational demanding since they often consist of many cells and many timesteps and are generally not capable of using the available computer resources efficiently. It is important to note that the number of cells and timesteps are not only determined by the resolution required to capture features of interest, but can also be directly imposed by the necessity to satisfy various numerical constraints (Oude Essink, 2003). Short longitudinal dispersion lengths, that are common in sedimentary basins such as the deltaic area of The Netherlands, might stress the grid Péclet condition, leading to small cell sizes and hence many cells. Furthermore, the occurrence of large model groundwater flow velocities, while satisfying the Courant–Friedrichs–Lewy condition, might result in small transport timesteps and hence many timesteps. Finally, the large inertia of variable-density groundwater flow systems makes that often large simulation times are needed to accurately estimate future groundwater salinities, e.g. for paleo-hydrogeological reconstruction of the fresh-saline groundwater distribution (Delsman et al., 2014). This makes transient variable-density groundwater flow and salt transport simulations very computationally challenging.

Distributed memory parallel computing (see e.g. Eijkhout et al., 2015) is a commonly used practice to significantly reduce computing time and make large memory usage possible. Common distributed memory parallel computers use a non-uniform memory access architecture (NUMA). In NUMA, the entire computational grid (memory) is first dis-

* Corresponding author.
E-mail address: jarno.verkaik@deltares.nl (J. Verkaik).

https://doi.org/10.1016/j.advwatres.2021.103976
Received 2 March 2021; Received in revised form 31 May 2021; Accepted 3 June 2021
Available online 7 June 2021

tributed (partitioned) over multiple (computer) nodes, each node having local main memory (RAM), and each node consists of one or more multi-core CPUs (processors). Then, the problem is solved simultaneously by multiple cores while exchanging data through a fast interconnection network using the message passing interface (MPI; Forum, 1994). This is in contrast with shared-memory parallel computing, where each processor can only access the same memory of a single computer, typically through OpenMP. However, parallel computers nowadays are heterogeneous systems with a mixed memory organization where memories are distributed across the nodes while multi-core processors share memory within the same node or CPU. Solving the flow and salt transport equations in parallel typically means that after discretization consecutively one or more linear systems of equations need to be solved by e.g. a Krylov subspace linear iterative solver, such as the conjugate gradient method (CG) for symmetric positive definite matrices or the bi-conjugate gradient stabilized method (Bi-CGSTAB) for general matrices (Barrett et al., 1995), accelerated by a suitable parallel preconditioner such as the additive Schwarz (block Jacobi) preconditioner or multigrid (Smith et al., 1996).

The focus in this paper is on parallelization of SEAWAT (Langevin et al., 2008), the most widely used public domain code developed by the U.S. Geological Survey for modeling 3D variable-density groundwater flow and salt transport by coupling MODFLOW (Harbaugh et al., 2000) and MT3DMS (Zheng and Wang, 1999). Since SEAWAT is based on MODFLOW, the worldwide leading code for groundwater modeling, it largely benefits from the MODFLOW ecosystem of a vast number of pre- and post-processing tools (Bakker et al., 2016) and well documented graphics user interfaces (Waterloo Hydrogeologic, 2021).

Several proprietary codes claim to be suitable for modeling parallel 3D variable-density groundwater flow, such as TOUGH (Jung et al., 2018; Pruess et al., 2011) following a distributed-memory approach using an additive Schwarz preconditioned Bi-CGSTAB Krylov solver from the Aztec solver library (Elmroth et al., 2001), HydroGeoSphere using shared-memory OpenMP (Hwang et al., 2014), FEFLOW (Diersch, 2002) using the algebraic multi-grid solver from the SAMG solver library, and d^3f (Schneider et al., 2012) using the distributed memory multigrid solvers from the UG solver solver library (Lang and Wittum, 2005). As far as we know, only for the latter d^3f code parallel results were published for a 3D variable-density groundwater model, reporting a speedup of 282 on 512 computational cores for a saltdome problem having 66 million of unknowns (Lang and Wittum, 2005).

MODFLOW and MT3DMS as standalone (non-coupled) codes were parallelized by several researchers. Regarding MODFLOW, (semi-)distributed-memory parallelization was applied by Schreuder (2005), where one (Parent) process was set responsible for reading all model input data and scattering this data to the other (Worker) processes. As a linear parallel solver the additive Schwarz preconditioned CG solver from the PETSc library (Balay et al., 2014) was applied. Cheng et al. (2014) applied the algebraic multi-grid preconditioned PCG solver from the JASMIN library (Mo et al., 2010) to speed up computations for a field flow problem located at Yanming Lake in China. Naff (2008) applied a (semi-)distributed-memory parallelization, where he applied an additive Schwarz preconditioned CG solver, while keeping the matrix assembly and input/output serial. Shared-memory parallelization of MODFLOW using OpenMP was done by Dong and Li (2009). Parallelization of MODFLOW using graphics processing units was done by Hughes and White (2013), and Ji et al. (2014). Regarding MT3DMS, shared-memory parallelization using OpenMP was done by Abdelaziz and Le (2014) and Huang et al. (2018).

To our knowledge, we are the first to parallelize SEAWAT. The reason for parallelizing SEAWAT is that we aim to push 3D groundwater flow and salt transport modeling to a next level for a large group of geohydrologists by making our code available as open source software as part of interactive MODeling Water Quality software (iMOD WQ; Deltares, 2020). Also from that point of view, we therefore believe that our code differs from proprietary codes like d^3f (Lang and Wittum, 2005). Our parallelization approach has a similarity with the parallelization of MODFLOW done by Schreuder (2005) and Naff (2008), although there are significant differences. First, we have also parallellized salt transport computations, hence MT3DMS. Second, compared to Schreuder (2005), our physical overlap supports flexible rows of cells to account for the Total Variation Diminishing (TVD) advection option. Our physical overlap also supports communication for evaluating the cross terms when using the full dispersion tensor option instead of lumping the cross terms to the righthand-side. Second, our parallellization approach is fully distributed and does not include any Parent-Worker mechanism for scattering input data. Parallel output is supported and effcient parallel input can be used by grid-wise direct access binary reads, making our implementation more suitable in case of many varying stress input data and output. Comparing our work to all the above mentioned parallellelization efforts of MODFLOW and MT3DMS, we believe that our fully distributed memory approach is the reason why we, as far as we know, are the first to report speedups for a very large model (~100 million of active cells). Third, our parallelization depends on the MPI software library only, which is generally straightforward to use on cross-platforms and can therefore be used by a wide range of hydrogeological modelers. We developed a linear parallel unstuctured solver and do not make use of PETSc solvers, because at the time of development, PETSc was very hard to use by modelers using Windows computers. Fourth, our partioning differs in the way how it deals with irregular model boundaries, where we apply the commonly used and more robust orthogonal recursive bisection partitioning (Berger and H. Bokhari, 1987).

It should be noted that regarding the salt transport advection scheme we did not parallelize the Lagarangian method, hence the method of characteristics (particle tracking). Supporting this would likely require the implementation of dynamic load balancing of computational work, involving re-partitioning of moving particles in time and dynamically re-mapping concentrations to the flow domain partitions. Since Eulerian methods suffer from numerical dispersion (Oude Essink and Boekelman, 1996; Zheng and Wang, 1999), our parallelization might therefore be less applicable to problems requiring the simulation of sharp interfaces between fresh and saline groundwater.

2. Methods

SEAWAT (Langevin and Guo, 2006) uses MODFLOW to solve the variable-density groundwater flow equation (see Eq. (3) in Appendix A) and MT3DMS to solve the salt transport equation (see Eq. (4) in Appendix A), coupled by the state-equation (see Eq. (6) in Appendix A), together with sufficient boundary and initial conditions. Both MODFLOW and MT3DMS apply a control-volume finite-difference (or finite volume) discretization on a structured grid, and typically require multiple so-called outer iterations (Picard linearization) to reach convergence, where in each outer iteration a linear system of equations is being solved using so-called inner iterations. Distributed memory parallelization of SEAWAT involves setting up the grid partitioning, modifying the linear solver, setting up the MPI communication, and optimizing the input and output. After setting the goals of the parallelization and how its performance is measured, we will address each of these aspects of the parallelization method in the next sections.

2.1. Parallel performance measurement

Since hydrogeological modelers typically like to speed up their existing model by using the available (processor) cores, we focus in our paper on obtaining strong parallel scalability. Measuring strong scaling means that the problem size is kept fixed while the number of processor cores being used increases. This contrasts with weak parallel scalability, where the problem size grows linearly with an increasing number of processor cores. Here, we express the parallel performance in (relative) speedup $S_p = T_{\text{ref}}/T_p$, with T_{ref} is the measured execution (wall-

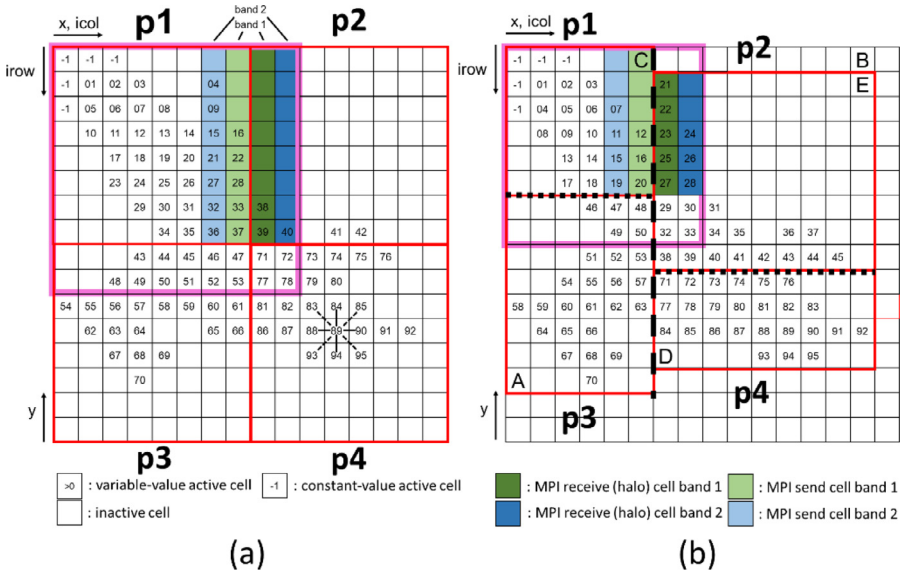


Fig. 1. Example of four partitions using two rows of overlap obtained by (a) uniform partitioning and (b) orthogonal recursive bisection assuming equal weights for the active cells. The red boxes denote the non-overlapping partitions; the pink boxes denote the overlapping partition for process p1. The blue and green cells denote the communication interface between process p1 and p2. Fig. 1b (k = 2): The first intermediate partition (load of 100) is determined by the minimum bounding box (BB) enclosing cells A and B. Since the longest dimension (15) is along the columns, a vertical cut is being made (black dashed line), resulting in two new, equally loaded, intermediate partitions (load 50; BB enclosing A and C; BB enclosing D and E). Since both intermediate partitions have the row direction as the longest dimension, vertical cuts (dotted black lines) are made, resulting in the final partitions (load of 25).

clock) time for a reference configuration with p_{ref} cores (or MPI processes/subdomains), and T_p is the measured execution runtime for an alternative configuration with p cores. In case the reference time corresponds to a serial run, then $T_{\text{ref}} = T_1$. In this definition of speedup, we assume that all measurements are obtained by applying the same solvers and solver settings, meaning that we do not seek for the fastest serial method and implementation. For ideal strong parallel scalability, the speedup equals $S_{p,\text{ideal}} = p/p_{\text{ref}}$. However, in practice this is hard to obtain due to (work)load imbalance, defined as $I = P\max_p(L_p/L)$, where P is the total number of cores being used, L_p is the work load for core p, and L the total work load. Partitioning primarily aims for obtaining a good load balance, and secondary for minimizing the subdomain interface length (edge-cuts).

2.2. Subdomain partitioning

Partitioning (or decomposing) here refers to dividing the computational grid consisting of active cells into smaller grids (partitions or subdomains), see Fig. 1 for an example of partitioning a grid consisting of 16 columns ($n_c = 16$) and 14 rows ($n_r = 14$) into four partitions ($P = 4$). Two methods are considered in this study: a relatively straightforward method to obtain equally sized rectangular partitions, here referred to as so-called uniform partitioning, and orthogonal recursive bisection (ORB) partitioning. Partitioning is done in the horizontal plane only, since for groundwater models the number of cells (rows, columns) in the horizontal plane is usually significantly larger than the number of cells in the vertical dimension (layers). However, this is simply an implementation choice and not a limitation of the method presented here. This also holds for the implementation choice that each partition is uniquely assigned to one processor core (MPI process), hence we assume that the total number of subdomains equals the total number of processor cores. Moreover, the partitioning is static and done only once as a pre-processing step. This neglects the spatio-temporal variation in computing time that might occur during simulation causing load imbalance, and hence a decrease in parallel performance. Since it turns out that for groundwater simulations most computing time is being spent in the linear solvers, and in our model active cells remain active, this seems to be a valid assumption for us. However, for models that have a highly varying stress input data or a highly changing spatial non-linearity including cell rewetting, dynamic partitioning might be more appropriate. In addition, our partitioning results in non-overlapping rectangular partitions, where the overlap is added after the partitioning without accounting for the work load. For this study, we assume that each active cell has the same weight. Furthermore, we assume that flow and salt transport share the same partition. This seems a valid assumption for us, since for our models flow and salt transport always share the same model boundaries. By this, we do see any need to explicitly parallelize the coupling between the flow and salt transport computational domain, therefore strongly simplifying implementation.

2.2.1. Uniform partitioning

For regular rectangular grids, so-called uniform partitioning is a straightforward method for minimizing edge cuts, hence the number of connections at the interface between the partitions, while ignoring the work load that is typically defined by the active cells. Let n be the minimum of n_c and n_r. From all possible combinations $P = P_c P_r$, where P_c and P_r are the number of blocks (subdomains) in column and row-direction, respectively, that specific combination is selected such that $P_i = \max\{1, \lfloor n/\sqrt{n_c n_r P^{-1}} \rfloor\}$, where $i = c$ when $n = n_c$ or $i = r$ otherwise. For the example of Fig. 1a, process p1 clearly has the largest number of active cells (42, enclosed by red colored box) and the load imbalance for the non-overlapping uniform partitioning is $I = 4 \times 42/100 = 1.68$.

2.2.2. Orthogonal recursive bisection partitioning

Real-world groundwater flow and salt transport models usually have irregular model boundaries, e.g. following coastlines or watershed boundaries to define an area of interest. This means that the computational load, typically represented by the active cells, can be scattered across the entire computational domain. Load balancing for such irregular boundaries is challenging and requires an appropriate partition strategy. This was also observed by Schreuder (2005), who developed an iterative partition method for a groundwater model with an irregular domain of the San Luis Valley, Colorado, USA. This method initially starts with a uniform partition and iteratively merges cell-weighted rectangles and shifts interfaces to obtain sub-optimal partitions. However, this method was found to be not sufficiently general or robust for general use (Schreuder, 2005).

As a more robust alternative method to account for irregular boundaries our partitioning supports orthogonal recursive bisection (Berger and H. Bokhari, 1987; Boman et al., 2012; Fox, 1988), or briefly ORB. This commonly used divide-and-conquer partitioning method recursively bisects intermediate partitions perpendicular to their longest dimension $k \geq 0$ times until $P = 2^k$ non-overlapping, equally loaded, partitions are obtained. For the example of Fig. 1b, all the non-overlapping partitions have the same load (equal to 25), and hence there is no load imbalance.

In the originally proposed method (Berger and H. Bokhari, 1987), the number of partitions can only be a power of two and therefore we modify this method to be applicable to general number of partitions P. This is done by applying a prime factorization of $P = \prod_{i=1}^{k} f_i$, where at each level i recursively f_i parts of equal load are obtained by dividing along the longest dimension. When $f_i = 2$ for all i, the original ORB method is obtained. It should be noted that our ORB implementation also includes the Dirichlet boundary conditions (or constant-value active cells; as in example Fig. 1b denoted by index -1). Since these cells are eliminated in the linear system, this means that a load imbalance might occur during linear solving. Although the ORB partitioning could therefore be further optimized for these boundary conditions, this was not a subject of this study.

2.2.3. Overlap and communication

Solving the variable-density groundwater flow equation and salt transport equation in parallel requires that all necessary coefficients at subdomain interfaces should be available for the discretization and evaluation of the computational stencil. For solving the variable-density groundwater flow equation, this means that interface coefficients such as the (inter-cell) transmissivity should be known and (matrix-vector) operations with the 7-point stencil across interfaces should be possible in parallel. For solving the advection-dispersion equation in parallel, two downstream nodal concentrations should be available at interfaces when applying the third-order TVD advection scheme using the ULTIMATE limiter (Leonard, 1988; Zheng and Wang, 1999). Furthermore, when applying dispersion using the full tensor (Scheidegger, 1961), all data near subdomain interfaces should be available for evaluating the dispersion cross terms for the 19-point stencil. Since we follow a distributed memory parallelization approach, and data at subdomain interfaces typically depend on both processes sharing the interface, data availability can be problematic. To overcome these difficulties, and to strongly simplify parallel implementation, we introduce a so-called physical overlap by specifying additional cells (for the example in Fig. 1 denoted by the pink boxes). These so-called halo (or ghost cells) are used to automatically compute the correct coefficients at subdomain interfaces from shared input data without any communication or reconstruction necessary. Besides this, using a physical overlap requires less modifications of the serial code.

In our parallelization, each process is responsible for computing the groundwater heads and concentrations for the cells in the non-overlapping partition exclusively. This means that for the additional halo cells only copies of computed heads and concentrations are stored that are received from neighboring processes. Hence, besides synchronized heads and concentrations, halo cells are similar to other non-halo cells. Synchronization is done by local, point-to-point, MPI communication. This synchronization is typically done after the linear Krylov solver has finished, in which halo cells are used to synchronize search directions at each (inner) iteration to account for matrix-vector products involving the computational stencil across the subdomain boundaries. For this purpose, we use a data structure that is arranged in communication bands. Each band has sending cells, corresponding to cells where the associated process is responsible for, and receiving cells, corresponding to the halo cells where data from adjacent processes are being received. Fig. 1 illustrates this for the case of two bands, corresponding to an overlap of two rows that is required for the TVD advection scheme. For the uniform partitioning in of Fig. 1a, process p1 has local communication with neighbors p2 and p3. Considering the first band, p1 sends computed values to p2 from column 8 and rows 1-8 (light green) and receives computed value from p1 in the halo cells of column 9, rows 1-8 (dark green). Note that almost half of these cells are inactive and have a user-defined no-data value. Nevertheless, in the chosen implementation, these inactive cells are also communicated and are therefore redundant. When dispersion is applied accounting for the full tensor, p1 also needs to communicate with p4. In that case, cell 71 and cells 72, 77 and 78 are also part of the halo cells for band 1 and 2, respectively. It should be noted that although our code has the flexibility to perform a point-to-point communication for only the outer band, in this study we always communicate for all bands at the same time. This means that when TVD is applied, local communication within the linear solver is for two bands although only values from the first band are used. Furthermore, as can be seen in Fig. 1b, ORB partitioning results in more local communication than uniform partitioning since p2 has at least three neighbors (p1, p3 and p4). This additional communication is the trade-off for obtaining optimal load balance.

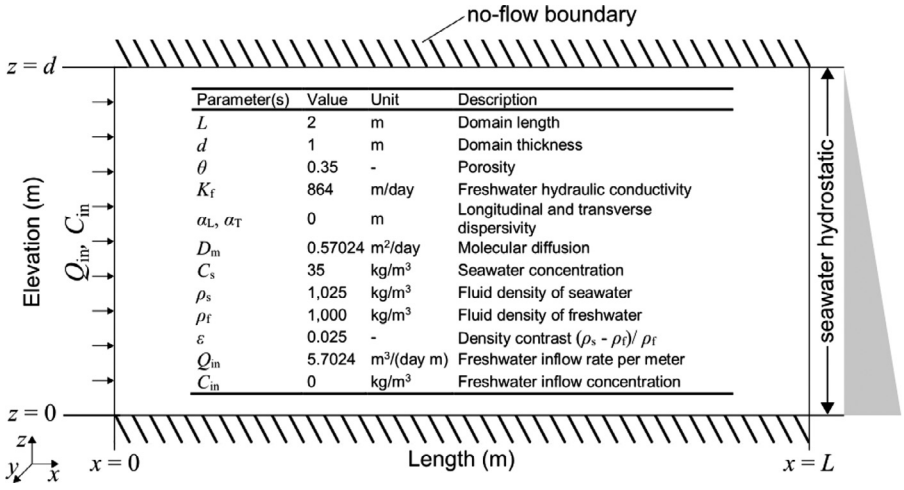
Besides local (point-to-point) communication, global (collective) MPI communication is required. This is mainly required within the linear Krylov solver, for computing inner (vector-vector) products and evaluating global grid maxima. Global communication is also necessary for synchronizing the salt transport timestep length (global grid minimum), accumulating volumetric or mass budgets for output, or gathering observation wells for output. In general, global communication is more expensive than local communication, since this type of communication requires a synchronization point involving all processes. For example, computing an inner (vector-vector) product value that necessarily needs to be available for each process prior to continuing to the next linear iteration, means that each process first needs to compute its partial sum before synchronizing, then sends this value to all other processes, and each process performs the addition. Global MPI communication strongly depends on the number of nodes used and the interconnection network characteristics such as diameter and latency (delay).

In this study, we mainly focus on relatively coarse-grained parallelization (many cells per subdomain) using fast interconnection networks having a low latency and high bandwidth. Coarse-grained means that the computational time, that can be directly related to partition size, is large compared to the total execution time. We therefore may neglect actual communication times spent on local and global communication. By this, we assume that communication overhead is exclusively the result of wait states, i.e. periods where processes sit idle at synchronization points, that can directly be related to load imbalance (Böhme, 2013)().

2.3. Linear parallel Krylov solver

The linear systems are solved by preconditioned Krylov subspace acceleration (see e.g., Barrett et al., 1995): for solving groundwater flow, the preconditioned CG method is used and for solving salt transport the preconditioned Bi-CGSTAB method. Within these methods, the (one-level) additive Schwarz preconditioner (Dolean et al., 2015; Smith et al., 1996) is taken, corresponding to the block matrix diagonal, see Appendix B for more details. Our implementation in SEAWAT, called the parallel Krylov solver (PKS), corresponds to a double precision unstructured grid solver that is largely based on the PCGU solver from MODFLOW-USG (Hughes and White, 2013). Commonly used Krylov subspace methods within SEAWAT are the PCG solver for MODFLOW (Hill, 1990) and the (compulsory) generalized conjugate gradient (GCG[1]) solver for MT3DMS (Zheng and Wang, 1999). In the remainder of this paper we will sometimes refer to those linear solvers as “default SEAWAT solvers”. Our PKS has the same termination criteria as the PCG and GCG solvers. Hence, for flow the infinity norm is used to declare converge of the linear iterates if the maximum absolute head differences are $\leq \varepsilon_{\text{hclose}}$ [m] and the maximum absolute head residuals are $\leq \varepsilon_{\text{rclose}}$ [kg/day] for sufficient small values of $\varepsilon_{\text{hclose}}$ [m] and $\varepsilon_{\text{rclose}}$ [kg/day]. For salt transport, the relative stopping criteria in the Euclidian norm is used for the concentrations satisfying small values of $\varepsilon_{\text{cclose}}$ [-]. For the remainder of this paper, we refer to the additive Schwarz preconditioned CG solver for solving flow as the flow PKS (FPKS) and the additive Schwarz preconditioned Bi-CGSTAB solver for solving salt

[1] Note the GCG solver being used in SEAWAT is also preconditioned, but this is omitted in the acronym.



Parameter(s)	Value	Unit	Description
L	2	m	Domain length
d	1	m	Domain thickness
θ	0.35	-	Porosity
K_f	864	m/day	Freshwater hydraulic conductivity
α_L, α_T	0	m	Longitudinal and transverse dispersivity
D_m	0.57024	m^2/day	Molecular diffusion
C_s	35	kg/m^3	Seawater concentration
ρ_s	1,025	kg/m^3	Fluid density of seawater
ρ_f	1,000	kg/m^3	Fluid density of freshwater
ε	0.025	-	Density contrast (ρ_s - ρ_f)/ ρ_f
Q_{in}	5.7024	m^3/(day m)	Freshwater inflow rate per meter
C_{in}	0	kg/m^3	Freshwater inflow concentration

Fig. 2. Boundary conditions and model parameters for the 3D Henry test case in the *xz*-plane.

transport as the transport PKS (TPKS). For both methods respectively, the algorithms are given by Fig. B1 and Fig. B2 in Appendix B, that include pseudocode for indicating the MPI communication points.

2.4. Input and output

Our parallelized code as part of iMOD WQ supports all ASCII and binary grid input data from standard SEAWAT in a serial manner, meaning that each process first needs to read the entire grid data into memory prior to clipping the data to its partition. Since this is not a scalable operation, such input mechanism might have negative effects on the parallel performance for transient simulations requiring many stress period input data. As a more scalable alternative, iMOD Data Files (IDFs; Vermeulen et al., 2019) are supported, where each process directly reads its local grid data exclusively by applying direct access binary reads. Standard SEAWAT column data input for stress packages, e.g. for river boundary conditions, are read in parallel by selectively reading only for cells that belong to the (overlapping) partition. As a more scalable option, we also support stress package input using IDFs by reading all parameters as grids. Output is done in a straightforward parallel manner, where each process uniquely writes its partition output, e.g. binary groundwater heads and concentrations. Besides standard SEAWAT files and IDFs, we support parallel VTK output files that can directly be visualized by graphical user interfaces such as ParaView and Tecplot. Furthermore, we extended the SEAWAT code by adding a pre-processing routine for setting up an entire SEAWAT model with a single key-word driven run file, that is easy-to-use and allows hydrogeological modelers to specify macros for efficiently setting up models consisting of many layers and stress periods.

3. Test cases

3.1. Test case 1: Henry 3D model

Solving the Henry saltwater intrusion problem (Henry, 1964) is a well-known benchmark for variable-density groundwater flow and salt transport simulation codes. The Henry test case considered here, see Fig. 2, depends on the three dimensionless parameters $a = Q_{in}/(K_f \varepsilon d)$, comparing viscous and buoyancy forces, $b = D_m/Q_{in}$, comparing diffusive and advective salt fluxes, and aspect ratio $\xi = L/d$. For our test we take $a = 0.263$, $b = 0.1$ and $\xi = 2$, corresponding to what Henry evaluated semi-analytically, and similar values as taken by Langevin et al. (2008).

To make the problem computationally challenging for parallel computing, we expand the original 2D geometry to 3D by taking 2 units (m) in the *y*-direction perpendicular to the *xz*-plane, hence specifying $Q_{in} = 11.405$ m^3/day. However, by doing this, the flow and salt transport remains 2D in the *xz*-plane. For this test case, $\Delta x = \Delta y = 0.002$m is taken and $\Delta z = 0.01$m, resulting in 100 million active computational cells (1000 columns × 1000 rows × 100 layers).

The cell-centered finite-difference scheme is taken for advection and for dispersion all cross terms are lumped to the righthand-side. Coupling between flow and salt transport is chosen to be explicit. For the FPKS the chosen stopping criteria are $\varepsilon_{hclose} = 1.0 \times 10^{-8}$ m and $\varepsilon_{rclose} = 1.0 \times 10^{-5}$ kg/day, for the TPKS $\varepsilon_{cclose} = 1.0 \times 10^{-7}$ [-]. Since the flow problem is linear in each time step, a maximum number of inner iterations (2500) for both linear solvers are taken such that convergence is achieved after one single outer iteration.

For this test case, a maximum of 1024 cores are taken considering a uniform partitioning. The number of subdomains in *x*- and *y*-direction is 1, 2, 4, 6, 8, 10, 14, 16, 24 and 32, and a single subdomain in z-direction. Hence, using 1024 cores this corresponds to a *xyz*-decomposition of $32 \times 32 \times 1$. Since for this test case it is assumed that there is no flow in *y*-direction, this means that the domain decomposition in *x*-direction contributes directly to the linear solver iterations, and the decomposition in *y*-direction corresponds to data decomposition only. All the grid input data for this test case are specified as binary IDF files (Vermeulen et al., 2019).

3.2. Test case 2: Sand Engine model

In 2011, a large sand nourishment of approximately 20 million m^3 was placed along the Dutch coastline between Hook of Holland and The Hague for testing its protective efficacy against long-term climate change effects such as sea level rise and storm surges (Mulder and Tonnon, 2011; Stive et al., 2013). As part of this Sand Engine pilot project, a SEAWAT model was developed (Huizer et al., 2016), here referred to as “Sand Engine model”, for assessing the effects of long-term morphological evolution of the sand replenishment on the local groundwater system and the existing fresh groundwater resources.

This calibrated SEAWAT model computes both variable-density groundwater flow and salt transport in 3D using 50 model layers, 234 rows, 234 columns, having a uniform horizontal cell size of 50 m with a varying layer thickness of 1 to 10 m, covering a total model area of 11.7 km^2. The molecular diffusion coefficient used is $10^{-9} m^2/s$, the longitudinal dispersivity used is 0.2 m, with a ratio of transversal to longitudinal dispersivity of 0.02. The model computes total dissolved solutes (TDS) values, with seawater density of $\rho_s = 1020$ kg/m^3 and concentration $C_s = 28$ kg/m^3, that is lower than the average salinity of the North Sea due to a freshening by the river Rhine. The chloride concentrations are obtained afterwards by multiplying the computed TDS values by a factor of 0.55.

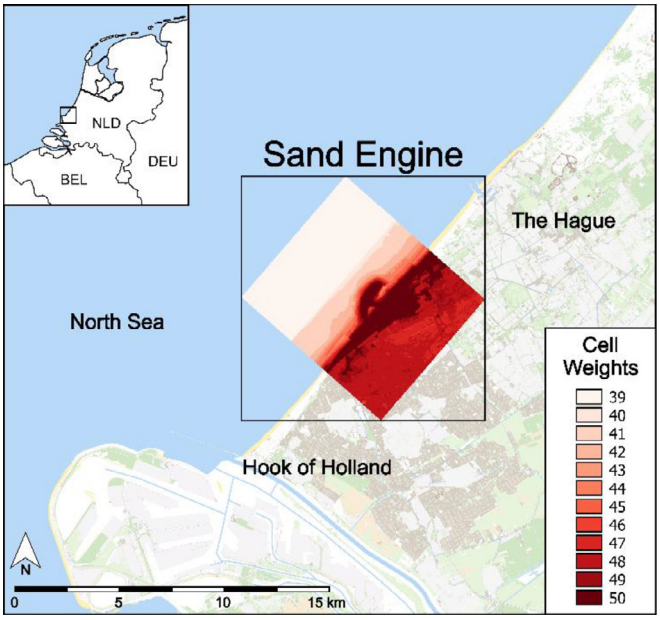


Fig. 3. Sand Engine model location and used cell weights for orthogonal recursive bisection partitioning.

In this paper we consider a realistic test case version of the Sand Engine model having a higher horizontal resolution of 25 m and simulating half a year using daily timesteps (July till December 2011). Having 25 m cells, this test case consists of a total of 4,733,797 active cells (50 layers, 468 rows, 468 columns). The TVD scheme and cell-centered discretization is used for advection, and for dispersion all cross terms are lumped to the righthand-side. Coupling between flow and salt transport is chosen to be explicit. For solving groundwater flow, the FPKS criteria are $\varepsilon_{\text{hclose}} = 1.0 \times 10^{-6}$ m and $\varepsilon_{\text{rclose}} = 0.1$ kg/day, setting 2500 as a maximum number of linear solver iterations. For solving salt transport, $\varepsilon_{\text{cclose}} = 0.1$ [-] is taken for the TPKS with a maximum of 50 iterations. An initial timestep of 1.0×10^{-3} days is taken and a total of 186 transport timesteps are required to finish the simulation.

For this test case, a maximum of 256 subdomains are used generated by ORB partitioning. Cell weights are defined as the sum of active cells in z-direction, see Fig. 3. All the grid input data for this test case are specified as standard SEAWAT ASCII data.

3.3. Hardware and compiler

The parallel performance of both test cases is evaluated on the Cartesius Dutch national supercomputer (SURFsara, 2014). At the time of writing, this machine has 1913 NUMA nodes connected by a fast Infiniband interconnection network in a fat tree topology, summing up to a total of 47,776 Intel Xeon CPU cores and 128 TB RAM memory. All our experiments are done on so-called (Haswell) thin nodes, where each node has 64 GB RAM memory and two Intel Xeon E5-2690 v3 12-core processors, and each CPU has its own socket and four channels to main memory. We use a maximum of four cores per thin node (two cores per CPU) since this value generally results in lowest wall-clock times. This is based on our experiences on Cartesius and a variation of cores per node for a fixed number of cores considering our test cases. This means that during computation at least 20 cores are idle and therefore the core utilization is low (17% non-idle). The reason why using more cores per CPU results in an increase in run times is very likely related to the large memory usage of our models and the competition of multiple cores within a multi-core CPU for the main memory bandwidth through the caches (Tudor et al., 2011). This seems a hardware related issue inherent to multi-core NUMA architectures and not a direct result from our parallelization approach, and therefore further research on core utilization is kept outside the scope of this study. Nevertheless, first observation for the largest Henry 3D case indicate that run times can increase ~15% when using four cores per CPU (33% non-idle) and >100% when using all 12 cores per CPU. Although hybrid MPI-OpenMP parallelization could possibly be a good approach to increase core utilization, previous experiments with MODFLOW did not show significant improvement (Verkaik et al., 2015). Furthermore, for each core configuration two runs are executed, and the minimum of wall-clock times is taken to exclude hardware variation, e.g. due to CPU throttling or network related issues.

We compiled iMOD WQ on Cartesius using the Intel Fortran compiler v15.0.0, using high level O3-optimization, linked with the Intel MPI library v5.0 update 3. We used mixed real-precision compilation, meaning that variables of type real used in the code can have single precision accuracy (4 bytes) or double precision accuracy (8 bytes). This contrasts with the USGS provided SEAWAT binary that was compiled such that each real variable corresponds to a double precision accuracy (compiler flags: -real-size 64, -align dcommons). The reasons for not doing this explicitly are the incompatibility with the iMOD library, that requires mixed precision, and the increasing memory usage that double precision usage involves. However, mixed real-precision might result in instability due to rounding errors with SEAWAT, although none of this behavior is observed for the test cases considered in this paper.

4. Results

4.1. Henry 3D test case

Fig. 4a shows the results for running the Henry 3D test case on the Cartesius supercomputer up to 1024 cores, using the PKS. The speedups are presented as relative values to two cores ($p_{\text{ref}} = 2$), since the serial run requires more memory (96 GB) than available on a single thin node (64 GB maximum). Because of this, we take two thin nodes for evaluating the speedup with two cores. Using 1024 cores, a maximum relative speedup of 140 is obtained. To estimate the value of the absolute speedup, a smaller run, using 10 layers, is compared on both a thin and so-called fat node that has more memory (256 GB) but a slower CPU (2.7 GHz Intel Xeon E5-4650). Estimating that the fat node is 1.13 times slower, the serial execution time on a thin node is estimated to be 16 hours 57 minutes 47 seconds, hence estimating the speedup with two cores to be $S_2 = 1.78$. This estimates the absolute speedup with 1024 cores to be $140 \times 1.78 \approx 249$. Fig. 4b shows the total number of linear iterations, hence accumulated over all outer iterations. It can be clearly seen that the total number of linear solver iterations increases with the number of processor cores used. For the FPKS iterations considering flow this is a factor 3, for the TPKS iterations considering salt transport this is a factor 3.7. The accuracy of the PKS is verified against the combined PCG (flow) and GCG (transport) solvers for a sample configuration of 144 cores, see Fig. 5 for the computed differences in isochlors.

4.2. Sand Engine test case

Fig. 6a shows the measured speedups and Fig. 6b the linear solver iterations, for running the Sand Engine test case on Cartesius up to 256 cores ($p_{\text{ref}} = 1$). Using 256 cores, a speedup of 86 is obtained. While the linear FPKS iterations for flow increased with a factor 1.7 compared to the serial run, the linear TPKS iterations for salt transport remain constant at a value of 372. For 256 cores using the PKS, Fig. 7a and c show the computed groundwater table and concentrations at the last time step, respectively, and differences (serial – parallel) compared to the serial run with the PCG and GCG solvers in Fig. 7b and d accordingly. For the head differences, the minimum, maximum and average values are -3.14×10^{-5} m, 1.24×10^{-5} m, and 2.41×10^{-6} m, respectively. For the concentrations those values are -5.40×10^{-3} mg/l, 7.34×10^{-3} mg/l and 2.41×10^{-6} mg/l, respectively.

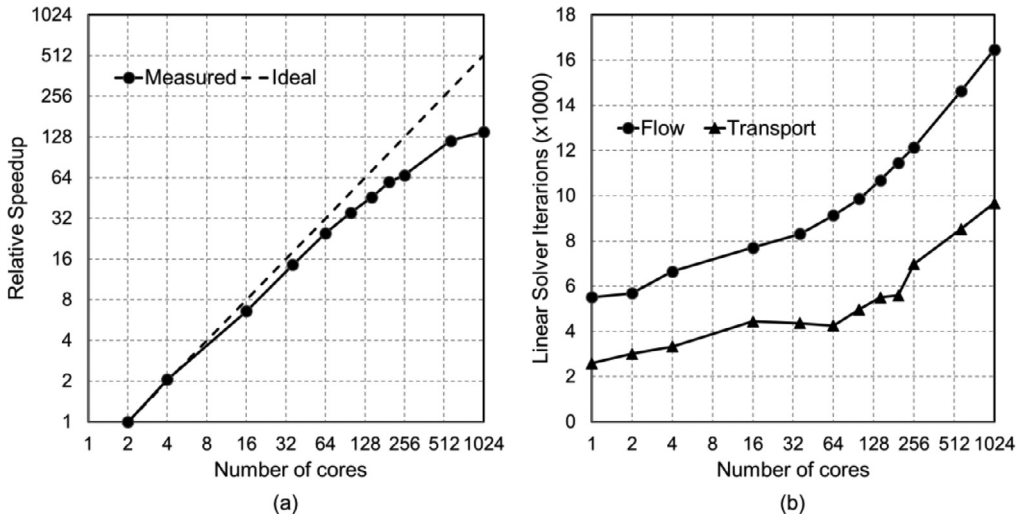


Fig. 4. Measured speedups relative to two cores (a) and linear solver iterations (b), considering the Henry 3D test case. The execution time was reduced from 9 hours, 31 minutes and 12 seconds (2 cores) to 4 minutes and 5 seconds (1024 cores).

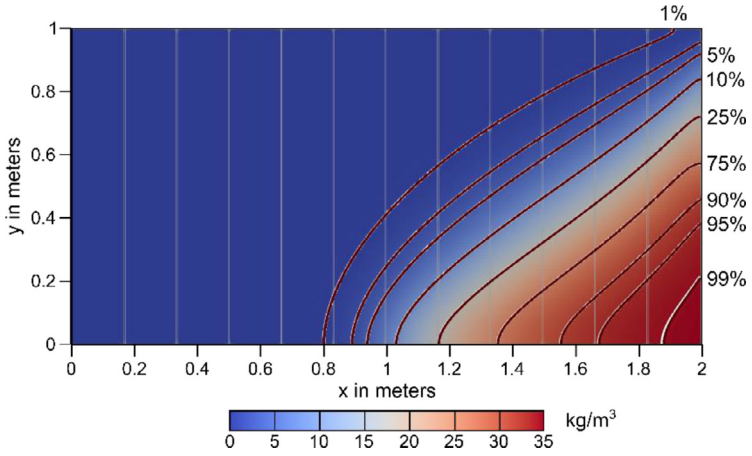


Fig. 5. Computed isochlors for the Henry 3D test case, relative to seawater concentration of 35 kg/m^3, comparing the PKS using 144 cores (black lines) to the combined serial PCG and GCG solver (white lines); it can be seen that both lines most of the time lie on top of each other.

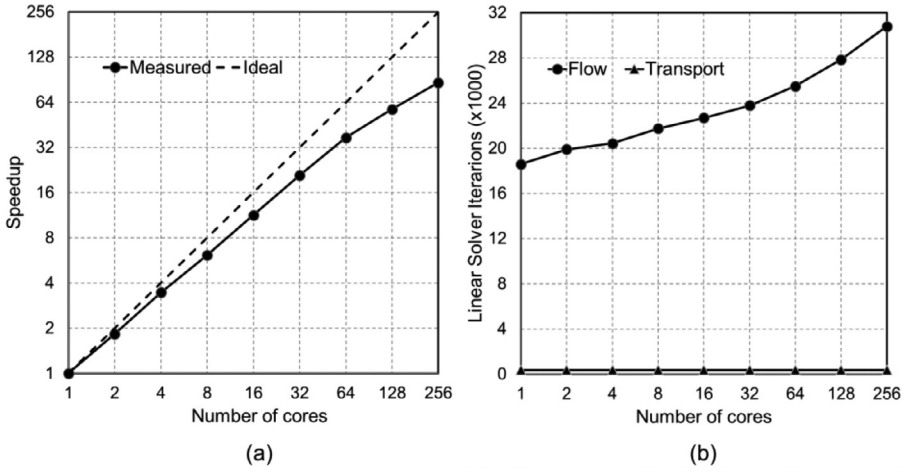


Fig. 6. Measured speedups (a) and linear solver iterations (b) for the Sand Engine test case. Using 256 cores, the execution time is reduced from 1 hour 47 minutes 55 seconds to 2 minutes 40 seconds.

5. Discussion

The results show that significant speedups are obtained (Figs. 4a and 6a), up to two orders of magnitude for the Henry 3D test case, and for both test cases there is a strong linear solver iteration increase (Figs. 4b and 6b) for an increasing number of processor cores. However, surprisingly, this iteration increase does not hold for the Sand Engine transport iterations that remain constant (see Fig. 6b). The reason for this might be found in the small mechanical dispersion used in the Sand Engine test case, implying highly advection dominated flow and salt transport that therefore does not heavily strain the linear salt transport solver.

It can be seen in Figs. 4a and 6a that for an increasing number of cores the speedup curves tend to deviate more from ideal speedups and therefore our parallelization becomes less scalable (see

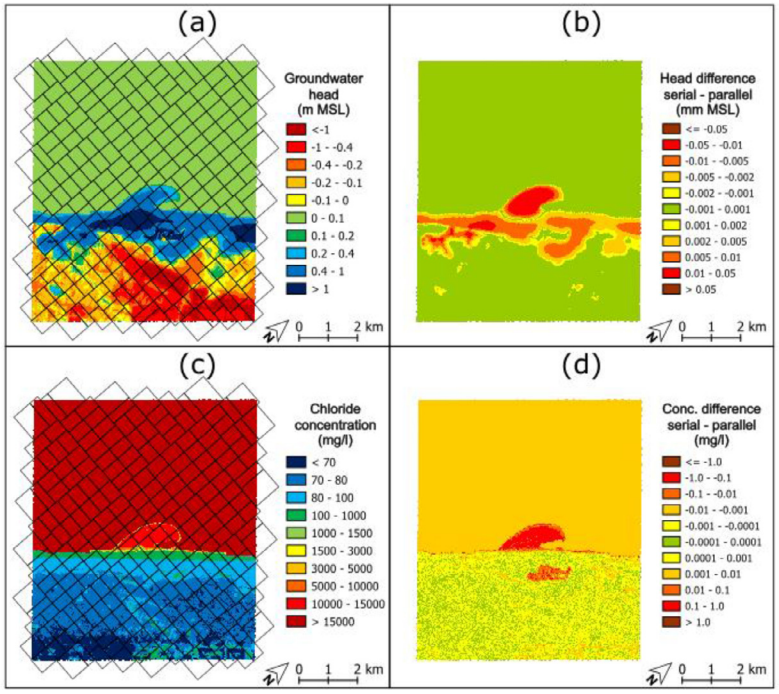


Fig. 7. Results for comparing a parallel run using 256 cores and the PKS with a serial run (PCG & GCG), considering output for January 1st of 2012 (time step 184). Subplot (a) and (b) show the groundwater heads computed in parallel (meters above Mean Sea Level) and the difference serial – parallel (in millimeters), respectively. Subplot (c) and (d) show the chloride concentrations computed in parallel (in milligrams per liter) and the difference serial – parallel (in milligrams per liter), respectively.

Table 1
Root mean squared errors of the computed groundwater heads (top value, in m) and computed salt concentrations (bottom, in TDS) for the Sand Engine test case comparing different solvers.

	PCG & GCG, $P = 1$	FPKS & TPKS, $P = 1$	PCG & TPKS, $P = 1$	FPKS & GCG, $P = 1$	FPKS & TPKS, $P = 256$
PCG & GCG $P = 1$	-	-	-	-	-
FPKS & TPKS $P = 1$	5.09×10^{-4}m 9.69×10^{-5}kg/m^3	-	-	-	-
PCG & TPKS $P = 1$	5.09×10^{-4}m 6.36×10^{-5}kg/m^3	9.73×10^{-7}m 1.35×10^{-5}kg/m^3	-	-	-
FPKS & GCG $P = 1$	1.10×10^{-6}m 9.28×10^{-6}kg/m^3	5.09×10^{-4}m 9.71×10^{-5}kg/m^3	5.09×10^{-4}m 1.48×10^{-5}kg/m^3	-	-
FPKS & TPKS $P = 256$	5.09×10^{-4}m 9.71×10^{-5}kg/m^3	9.95×10^{-7}m 1.31×10^{-5}kg/m^3	1.08×10^{-6}m 9.69×10^{-5}kg/m^3	5.09×10^{-4}m 9.70×10^{-5}kg/m^3	-

Section 2.1). We analyze this behavior by using Scalasca code profiling tools (Geimer et al., 2010) to identify the main non-scalable components of the code. This is done by analyzing the component cost $C_p^c = pT_p^c$ for a component c in relation to the total cost $C_p = pT_p$. Fig. 8a and c show C_p^c/C_{ref} for the Henry 3D test case ($C_{\text{ref}} = C_2 = 2T_2$) and Sand Engine test case ($C_{\text{ref}} = T_1$), respectively. In Fig. 8b and d values for C_p^c/C_p are plotted for the Henry test case and Sand Engine test case, respectively, showing the portion of work for a certain component relative to the total execution time. In the ideal case, perfect strong scaling (or cost optimality) is achieved if for all (or at least for the most dominant) components C_p^c/C_{ref} remains constant for an increasing number of cores. However, as can be seen in Fig. 8a and c this is not the case.

The linear solver execution times are the main non-scalable components that increase with the number of processes (see FPKS and TPKS in Fig. 8a and c). The reason for this is the linear solver iteration increase (see Figs. 4b and 6b), that can likely be explained from mathematics by the presence of low frequency eigenmodes induced by the domain decomposition in the dominant flow direction, hampering the linear solver convergence (Dolean et al., 2015; Smith et al., 1996). This is a known problem (see e.g. Hammond et al., 2014) and our solver convergence can likely be improved by combining the additive Schwarz preconditioner with a multi-level preconditioner, such as the additive coarse grid correction preconditioner. The Henry case would likely benefit most from applying such preconditioner, since up to 256 cores the portion of work spend in the linear solver remains constant at about 80% (see in Fig. 8b by adding for FPKS and TPKS). We can estimate a theoretical upper bound for the speedup by assuming no iteration increase, as shown in Fig. 9. The speedups are estimated by using the linear iteration time (computation and MPI; cost of ~5 and ~9 seconds using 2 cores, for FPKS and TPKS respectively) and reducing the total execution time linearly with the iteration increase, while neglecting any additional work introduced by multi-level preconditioning. By doing this, almost perfect scalability can be achieved up to 256 cores. Correcting for iteration loss we estimate $S_{256} = 602$ which is big improvement of a factor 2.4. This, more or less, fine tuning of our parallelization is recommended for future research.

Considering the Henry test case, load imbalance occurs when using more than 256 cores. This can be seen in Fig. 8b by the downward trend of the linear solver computation time and upward trend of MPI communication time. Load imbalance is likely to be caused by the physical overlap, that is added after the partitioning (see Section 2.2). For the Henry case this is one additional row for interfacing with a neighboring subdomain since we are using finite-difference advection. This means that the amount of total work will increase, e.g. the time spend on matrix assembly for dispersion, see the slightly upward trend of "T Mat. Assembly" in Fig. 8a starting from 256 cores. Although the linear

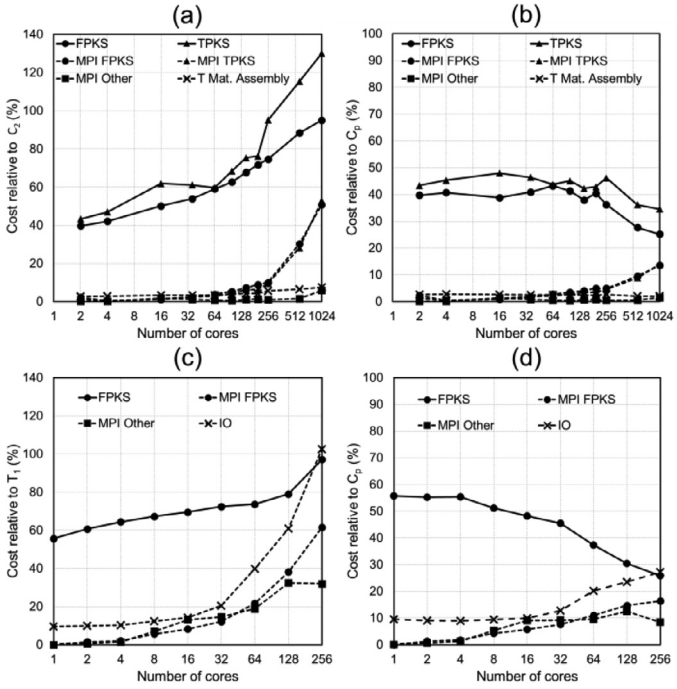


Fig. 8. Measured time for the non-scalable components, presented as component costs relative to reference cost, C_p^c/C_2 for the Henry test case (a), and C_p^c/T_1 for the Sand Engine test case (c), and component cost relative to parallel costC_p^c/C_p for the Henry test case (b) and Sand Engine test case (d). "FPKS": execution time within the flow PKS; "TPKS": execution time within the salt transport PKS; "MPI FPKS": communication wait time for the flow PKS; "MPI TPKS": communication wait time for the salt transport PKS; "MPI other": communication wait time other than within solvers; "T Mat assembly": matrix assembly execution time for salt transport; "IO": input/output time.

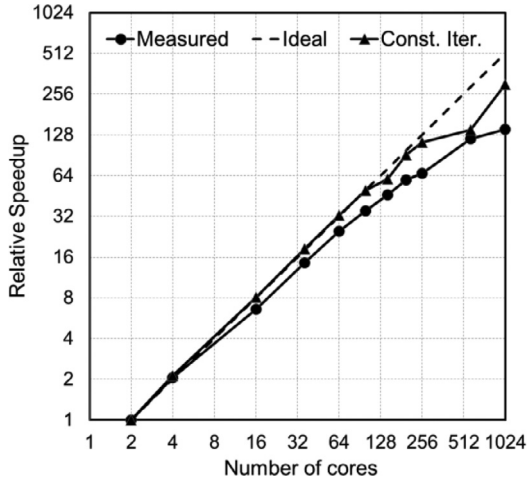


Fig. 9. Estimated relative speedups for the Henry 3D test case assuming no iteration increase.

solvers are mathematically strict non-overlapping, we also expect that the linear solver is responsible for a certain part of the load imbalance because all DO-loops in our code are defined over all the cells including the overlap.

The reason why larger speedups are obtained for the Henry test case (Fig. 6a) than for the Sand Engine test case (Fig. 7a), might be directly related to the partition sizes and hence the grain sizes. For each core configuration, the number of active cells for each partition is about 21 times larger for the Henry test case than for the Sand Engine test case, making the parallelization coarser grained. Considering 256 cores, the speedup for the Sand Engine model of 86 (~19.5k active cells per subdomain) is smaller than the estimated speedup of 119 (= 66.6 × 1.78; see Section 4.1) for the Henry test case (~390k active cells per partition). As long as the computational time remains dominant, one should only have to focus on parallelizing computation, which is generally relatively straightforward to do in order to obtain strong scaling as we have illustrated for the iteration increase. The effect of the finer-grained parallelization for the Sand Engine can be seen in Fig. 8c, where the MPI curves grow faster than the linear solver iteration computational time, indicating load imbalance. Furthermore, for the Sand Engine test case I/O becomes dominant taking 27% of the total execution time when using 256 cores. The non-scalability of I/O for the Sand Engine test case can be explained by the usage of standard SEAWAT input data that does not scale in parallel (see Section 2.4). We expect to improve this by using iMOD binary data files (IDFs) exclusively for this model in the future. Since the Sand Engine test case is relatively fine-grained, analysis of constant iteration increase did not show any significant improvement in speedups because of the load imbalance and I/O overhead.

As a rule of thumb for users, one can say that our parallelization is most suitable for relatively larger models (>> 10^5 active cells per partition), lesser suitable for relatively smaller models (10^4 – 10^5 of active cells per partition), and not suitable for very small models (< 10^4 of active cells per partition). Although highest speedups are to be expected for relatively larger models like the Henry 3D test case, still significant (although less efficient) speedups can be obtained when a sufficient number of processor cores is available, as illustrated by the Sand Engine test case.

From our modeling experiences we believe that our PKS computes sufficiently accurate heads and concentrations that have comparable accuracy compared to the default PCG and GCG solvers in SEAWAT. Nevertheless, from a numerical point of view, a hydrogeological modeler should be aware that small differences might occur. This is also the case for the Sand Engine test case, see Fig. 7, where the observed maximum absolute head difference of 3.14×10^{-5} m for the top view layer with active cells is larger than $\varepsilon_{\text{hclose}} = 1.0 \times 10^{-6}$, as well as for the Henry test case, where small differences occur for the 99% isochlors, see Fig. 5. Focusing on the Sand Engine test case, we also observe that at the final timestep, some computational cells have higher absolute maximum differences (3.04×10^{-4} m for head and 4.84×10^{-1} kg/m^3 TDS) than the solver stopping criteria. Although we did not find motivations for doing an extensive accuracy analysis, we suspect that the reasons for these small differences that accumulate during simulation time are likely to be related to differences in solver methods, rounding errors due to finite precision arithmetic, and perhaps too aggressive compiler optimization. The preconditioner used in the PKS (ILU(0)) differs from the one used within the PCG solver (modified incomplete Cholesky preconditioner) and the one used within the GCG solver (symmetric successive over relaxation). Furthermore, the FPKS used the Bi-CGSTAB Krylov method while the GCG solver uses Lanczos/ORTHOMIN. These differences in solver methods might result in differences in accuracy after solving the linear systems for flow and salt transport. Furthermore, mathematically seen, for each different core configuration a different additive Schwarz preconditioner is applied with the PKS, hence resulting in different convergence behavior. Moreover, different core configurations result in different finite-precision, rounded arithmetic, e.g. for the computing of inner product coefficient within the CG algorithm by adding partial sums that are computed by each core simultaneously. In addition, the TPKS has double precision accuracy while the GCG solver has single precision accuracy, meaning that in our applications using the GCG solver might introduce small rounding errors. Table 1 shows the root mean square error (RMSE) measured over all daily timesteps and cells of the Sand Engine test case, comparing the PKS (FPKS & TPKS) for 256 cores with different serial solver configurations (default SEAWAT solvers PCG & GCG, FPKS & TPKS, PCG & TPKS, and FPKS & GCG). The RMSE values show that the main difference between the PKS using 256 cores and the default SEAWAT solvers (see also Fig. 7) is caused by the differ-

ence between the GCG solver and the TPKS. This is very likely related to differences in solver methods or finite precision arithmetic.

6. Conclusions and recommendations

A parallel version of SEAWAT (MODFLOW/MT3DMS) for simulating 3D variable-density groundwater flow and salt transport is developed, as part of iMOD WQ. In this paper, we showed that our parallel implementation is capable of significantly reducing computing times, up to two orders of magnitude, and capable of running models with very large memory requirements. Our parallelization differs from other MODFLOW and MT3DMS parallelization efforts done by other researchers in a way that a) salt transport is parallelized using a physical overlap to account for e.g. TVD advection; b) our approach fully distributes memory including parallel input/output; c) we apply the robust orthogonal recursive bisection partitioning method to address irregular model boundaries. Since our parallelization is easy to use and open source, we expect that our parallelization may lead to new scientific advancement in the field of groundwater flow and salt transport modeling, where ultimately huge 3D problems having billion of cells can be solved with thousands of cores. Therefore, we believe that our work can be an asset for water managers and decision makers in coastal areas, in a way that it helps to provide them with high-resolution estimates and future projections of the groundwater salinity distribution and fresh groundwater reserves.

Although our reported speedups are high, there is still room for future improvement. First, our parallelization may be improved by reducing the linear solver iteration increase by applying a suitable parallel multi-level preconditioner. Second, load balancing might by further optimized by explicitly considering the weights of the physical overlap during the partitioning phase. Third, our implementation should be further analyzed on different hardware and multi-core CPU configurations with the aim of improving processor core utilization. We expect that in the future multi-core CPUs, having more and more cores, will become more efficient for memory-bound problems (such as the latest generation AMD EPYC Zen CPUs) and result in improved speedup and better core utilization for 3D variable-density groundwater flow and salt transport modeling.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

J. Verkaik: Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data curtion, Writing – original draft, Writing – review & editing. **J. van Engelen:** Resources, Writing – review & editing. **S. Huizer:** Methodology, Writing – review & editing. **M.F.P. Bierkens:** Conceptualization, Writing – review & editing. **H.X. Lin:** Conceptualization, Writing – review & editing. **G.H.P. Oude Essink:** Conceptualization, Writing – review & editing.

Acknowledgements

We thank Christian D. Langevin and Joseph D. Hughes for their suggestions and comments on parallelization SEAWAT. We also thank Deltares and Utrecht University for making this research possible. This work was part of the development of iMOD WQ (https://oss.deltares.nl/nl/web/imod) and we thank Gijs Janssen for his support. Furthermore, we thank Edwin Sutanudjaja and Martijn Russcher for their support on running jobs on Cartesius. All experiments were carried out on the Dutch national e-infrastructure with the support of the SURF Cooperative. The authors also would like to thank the editor and two anonymous reviewers for their valuable comments and suggestions.

A. Governing equations for variable-density groundwater flow and salt transport

Conservation of fluid mass can be expressed by the continuity equation (see e.g. in Bear, (1979))

$$\frac{\partial(\theta\rho)}{\partial t} + \nabla\cdot(\rho\mathbf{q}) - \rho_{ss}q_{ss} = 0, \tag{1}$$

with θ [-] the effective porosity, ρ [kg/m^3] the groundwater density, $\mathbf{q}$ [m/day] the specific discharge (or Darcy velocity) vector, ρ_{ss} [kg/m^3]and q_{ss} [1/day] the fluid density and flow rate from the sinks and sources, respectively. Following the notation of Guo and Langevin (2002), conservation of momentum can be expressed by Darcy's law, neglecting viscosity effects for sake of simplicity,

$$\mathbf{q} = -\mathbf{K}_f(\nabla h_f + \frac{\rho-\rho_f}{\rho_f}\nabla z), \tag{2}$$

where $\mathbf{K}_f$ [m/day] is the hydraulic conductivity tensor for saturated fresh groundwater, h_f [m] the (to-be-solved) fresh groundwater head, ρ_f [kg/m^3] the freshwater density (typically 1000 kg/m^3) and z [m] the elevation. Assuming isothermal conditions, expanding $\partial(\theta\rho)/\partial t$ and substituting (2), Eq. (1) can be written as

$$\rho S_f\frac{\partial h_f}{\partial t} - \nabla\cdot\left[\rho\mathbf{K}_f(\nabla h_f + \frac{\rho-\rho_f}{\rho_f}\nabla z)\right] + \theta\frac{\partial\rho}{\partial C}\frac{\partial C}{\partial t} - \rho_{ss}q_{ss} = 0, \tag{3}$$

where S_f [1/day] is the specific storage for fresh groundwater and C [kg/m^3] is the salt (e.g. chloride) concentration to be solved. This equation is known as the variable-density groundwater flow equation (VDGFE), which reduces to the ordinary constant-density groundwater flow equation when $\rho = \rho_{ss} = \rho_f$. Neglecting chemical reactions, assuming that diffusive and dispersive salt transport are based on Fick's law, assuming single species, conservation of salt can be expressed by the governing advection-dispersion equation,

$$\frac{\partial(\theta C)}{\partial t} + \nabla\cdot(\mathbf{q}C - \theta\mathbf{D}\nabla C) - q_{ss}C_{ss} = 0, \tag{4}$$

where C_{ss} [kg/m^3] is the concentration for sink and sources, and $\mathbf{D}$ [m^2/day] the hydrodynamic dispersion tensor with coefficients, for incorporating mechanical dispersion and molecular diffusion. This equation can be expressed as

$$D_{ij} = (\frac{\alpha_T}{\theta}\|\mathbf{q}\|_2 + D_m)\delta_{ij} + \frac{\alpha_L-\alpha_T}{\theta}\frac{q_iq_j}{\|\mathbf{q}\|_2}, \tag{5}$$

where α_L [m] and α_T [m] are the and longitudinal and transversal dispersivities, respectively, D_m [m^2/day] the effective molecular diffusion coefficient, and δ_{ij} denotes the Kronecker delta. Eq. (4) is also known as the salt transport equation (STE; see Zheng and Wang, 1999). Since with the variable-density groundwater flow Eq. (3) and salt transport Eq. (4) we have three unknowns (h_f, ρ and C) an additional equation of state is required. Neglecting other density effects due to temperature and pressure, we can use the linear relation

$$\rho(C) = \rho_f + \frac{\partial\rho}{\partial C}C \approx \rho_f + \frac{\rho_s-\rho_f}{C_s}C, \tag{6}$$

where ρ_s [kg/m^3] and C_s [kg/m^3] are the density and salt concentration of seawater, respectively. Typical values for seawater are $\rho_s = 1025$ kg/m^3 and $C_s = 35$ kg/m^3 of total dissolved solutes simplifying Eq. 6 to $\rho(C) \approx 1000 + 0.7143C$.

B. Parallel linear solver algorithms for flow and salt transport

Finite volume discretization of the VDGFE Eq. (3) and STE Eq. (4) excluding advection, results in, after (Picard) linearization and eliminating the Dirichlet boundary (constant-value) conditions, the linear equation

$$\mathbf{A}_{\{f,t\}}\mathbf{x}_{\{f,t\}} = \mathbf{b}_{\{f,t\}}, \tag{7}$$

where subscript *f* here denotes *flow*, and subscript *t* denotes *transport*. For solving the flow equation, $\mathbf{x}_f$ [m] is the vector of unknown fresh-water

$\mathbf{x}^{(0)}$ initial guess; $\mathbf{x} \leftarrow \mathbf{x}^{(0)}$; (1); $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$; ▷ (1): MPI local exchange of $\mathbf{x}$
for $k = 1, 2, \ldots,$ maxinner **do**
  $\mathbf{z} \leftarrow \mathbf{M}_{\mathrm{AS}}^{-1}\mathbf{r}$; ▷ Apply additive Schwarz preconditioner
  $\rho \leftarrow \mathbf{r}^T\mathbf{z}$; (2); ▷ (2): MPI global sum of ρ
  if $k = 1$
    $\mathbf{p} \leftarrow \mathbf{z}$;
  else
    $\beta \leftarrow \rho / \rho_0$;
    $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$;
  endif
  (3); $\mathbf{q} \leftarrow \mathbf{A}\mathbf{p}$; ▷ (3): MPI local exchange of $\mathbf{p}$
  $\gamma \leftarrow \mathbf{p}^T\mathbf{q}$; (4); $\alpha \leftarrow \rho / \gamma$; ▷ (4): MPI global sum of γ
  $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$; $\delta_x \leftarrow \|\alpha\mathbf{p}\|_\infty$
  $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{q}$; $\delta_r = \|\alpha\mathbf{p}\|_\infty$; (5); ▷ (5): MPI global max of both δ_x and δ_r
  if $\delta_x \le \varepsilon_{\mathrm{hclose}}$ **and** $\delta_r \le \varepsilon_{\mathrm{rclose}}$ **then stop**;
  $\rho_0 \leftarrow \rho$;
end

Fig. B1. FPKS algorithm for solving groundwater flow corresponding to the additive Schwarz preconditioned conjugate gradient algorithm. The symbol ← denotes that the left-hand side is assigned to the value of the right-hand side, according to Smith et al. (1996). "maxinner" is the maximum of inner iterations; for further notation see Fig. 2.5 of Barrett et al, (1995). The numbers (.) denote the MPI communication points.

heads, $\mathbf{A}_f$ [kg/(m • day)] a square, symmetric positive-definite, coefficient matrix with the hydraulic cell-by-cell conductivity times the density, and $\mathbf{b}_f$ [kg/day] the vector with groundwater sink/source and storage terms. The corresponding computational stencil is 7-point, hence $\mathbf{A}_f$ has 7 bands. For solving the transport equation, $\mathbf{x}_t$ [kg/m^3] is the vector of unknown salt concentrations, $\mathbf{A}_t$ [m^3/day] represents the discretization of dispersion/sink/source and is generally a nonsymmetric matrix, and $\mathbf{b}_t$ [kg/day] is the vector with sink/source terms. When full tensor dispersion is used, this will result in a 19-point stencil. However, in our study we assume that all dispersion cross terms are lumped to the right-hand side vector, resulting in $\mathbf{A}_t$ to have 7 bands like $\mathbf{A}_f$. For the preconditioned CG method (FPKS), the symmetrized preconditioned system

$$(\mathbf{M}^{-1/2}\mathbf{A}_f\mathbf{M}^{-1/2})\mathbf{M}^{1/2}\mathbf{x}_f = \mathbf{M}^{-1/2}\mathbf{b}_f, \quad \mathbf{M}^{-1/2}\mathbf{M}^{-1/2} = \mathbf{M}^{-1} \tag{8}$$

is solved and for the preconditioned Bi-CGSTAB (TPKS) the left preconditioned system

$$\mathbf{M}^{-1}\mathbf{A}_t\mathbf{x}_t = \mathbf{M}^{-1}\mathbf{b}_t, \tag{9}$$

where the matrix $\mathbf{M}$ is the preconditioner (Barrett et al., 1995).

Using block-wise natural node ordering for the non-overlapping partitions, as illustrated by the positive numbering in Fig. 1, the matrix $\mathbf{A}$ can be written as a block matrix of the form:

$$\begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,P} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ \mathbf{A}_{P,1} & \cdots & \cdots & \mathbf{A}_{P,P} \end{bmatrix}, \tag{10}$$

where $\mathbf{A}_{ii}$ correspond to the interior node coefficients and $\mathbf{A}_{i,j}$, $i \neq j$ to the coupling coefficients between the subdomains. Considering a 7-point computational stencil and a single band for the uniform partitioning example in Fig. 1a then block matrix (10) has 4×4 blocks ($P = 4$) and for the first subdomain p1 the interior coefficient sub-matrix $\mathbf{A}_{1,1}$ has dimension 37×37, local coupling sub-matrix $\mathbf{A}_{1,2}$ contains two non-zero entries (33 → 38, 37 → 39) and $\mathbf{A}_{1,3}$ four non-zero entries (34 → 44, 35 → 45, 36 → 46, 37 → 47), and $\mathbf{A}_{1,4} = \emptyset$. Note that in a distributed memory parallel setting block matrix (10) is never been formed explicitly since each processor only has local coefficients corresponding to a block row of this system.

Taking $\mathbf{M}$ as the block diagonal matrix of $\mathbf{A}$ results in the (non-overlapping) additive Schwarz preconditioner (Dolean et al., 2015; Smith et al., 1996), here denoted by $\mathbf{M}_{\mathrm{AS}}$:

$$\mathbf{M}_{\mathrm{AS}} \equiv \begin{bmatrix} \mathbf{A}_{1,1} & & & \\ & \mathbf{A}_{2,2} & & \\ & & \ddots & \\ & & & \mathbf{A}_{P,P} \end{bmatrix}.$$

In each CG or Bi-CGSTAB iteration, called inner iteration, the preconditioner is being applied (once for CG and twice for Bi-CGSTAB) and the system of the form $\mathbf{M}_{\mathrm{AS}}\mathbf{y} = \mathbf{z}$ has to be solved, where $\mathbf{y}$ and $\mathbf{z}$ are denoted as typical search directions. The benefit of the additive Schwarz preconditioner is that this can be solved entirely in parallel where each processor solves $\mathbf{A}_{i,i}\mathbf{y}_i = \mathbf{z}_i$ independently, called the local subdomain solve. Since it turns out that this can be done inaccurately, we apply an incomplete LU factorization with zero fill in (ILU(0)) using Gaussian elimination, where $\mathbf{A}_{i,i} = \mathbf{LU}$ for the non-zero entries of $\mathbf{A}_{i,i}$. The subdomain solution is obtained by first solving for the lower triangular matrix $\mathbf{L}\tilde{\mathbf{y}} = \mathbf{z}_i$ and then for the upper triangular matrix $\mathbf{U}\mathbf{y}_i = \tilde{\mathbf{y}}$. Convergence at the k-th inner iteration is obtained for FPKS by using the infinity norm ($\|\mathbf{y}\|_\infty \equiv \max_i |y_i|$) such that the stopping criteria $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_\infty \le \varepsilon_{\mathrm{hclose}}$ and $\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\|_\infty \le \varepsilon_{\mathrm{rclose}}$ are satisfied, for sufficiently small values of $\varepsilon_{\mathrm{hclose}}$ [m] and $\varepsilon_{\mathrm{rclose}}$ [kg/m^3]. For TPKS the criterium $\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\|_2 \le \varepsilon_{\mathrm{cclose}} \bullet \|\mathbf{b}\|_2$ should hold for Euclidian norm ($\|\mathbf{y}\|_2 \equiv (\sum_i y_i^2)^{1/2}$) for $\varepsilon_{\mathrm{cclose}}$ [-] small enough.

The FPKS and TPKS algorithms are given by Fig. B1 and Fig. B2, that include pseudocode for indicating the MPI communication points. Parallelization of these method involves a) local MPI point-to-point communication of vectors between subdomains prior to sparse matrix vector

1: $\mathbf{x}^{(0)}$ init. guess; $\mathbf{x} \leftarrow \mathbf{x}^{(0)}$; (1); $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$; $\tilde{\mathbf{r}} \leftarrow \mathbf{r}$; ▷ (1): MPI local exchange of $\mathbf{x}$
2: $\nu_b \leftarrow \mathbf{b}^T\mathbf{b}$; (2); $\nu_b \leftarrow \sqrt{\nu_b}$ ▷ (2): MPI global sum of ν_b
3: **for** $k = 1,\ldots,$ maxinner **do**
4: $\rho \leftarrow \tilde{\mathbf{r}}^T\mathbf{r}$; (3); ▷ (3): MPI global sum of ρ
5: **if** $k = 1$
6: $\mathbf{p} \leftarrow \mathbf{r}$;
7: **else**
8: $\beta \leftarrow (\rho / \rho_0) / (\alpha / \omega)$;
9: $\mathbf{p} \leftarrow \mathbf{p} - \omega\mathbf{v}$; $\mathbf{p} \leftarrow \mathbf{r} + \beta\mathbf{p}$;
10: **endif**
11: $\hat{\mathbf{p}} \leftarrow \mathbf{M}_{\mathrm{AS}}^{-1}\mathbf{p}$; ▷ Apply additive Schwarz preconditioner
12: (4); $\mathbf{v} \leftarrow \mathbf{A}\hat{\mathbf{p}}$; ▷ (4): MPI local exchange of $\hat{\mathbf{p}}$
13: $\gamma \leftarrow \tilde{\mathbf{r}}^T\mathbf{v}$; (5); $\alpha \leftarrow \rho / \gamma$; ▷ (5): MPI global sum of γ
14: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{v}$; $\delta_r \leftarrow \|\mathbf{r}\|_\infty$; (6); ▷ (6): MPI global max of δ_r
15: **if** $\delta_r \le \varepsilon_{\mathrm{mach}}$ **then set** $\mathbf{x} \leftarrow \mathbf{x} + \alpha\hat{\mathbf{p}}$ **and stop;** ▷ $\varepsilon_{\mathrm{mach}}$ is machine precision
16: $\hat{\mathbf{s}} \leftarrow \mathbf{M}_{AS}^{-1}\mathbf{r}$; ▷ Apply additive Schwarz preconditioner
17: (7); $\mathbf{w} \leftarrow \mathbf{A}\hat{\mathbf{s}}$ ▷ (7): MPI local exchange of $\hat{\mathbf{s}}$
18: $\omega_0 \leftarrow \mathbf{w}^T\mathbf{r}$; $\omega_1 \leftarrow \mathbf{w}^T\mathbf{w}$; (8); $\omega \leftarrow \omega_0 / \omega_1$; ▷ (8): MPI global sum of both ω_0 and ω_1
19: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\hat{\mathbf{p}}$; $\mathbf{x} \leftarrow \mathbf{x} + \omega\hat{\mathbf{s}}$;
20: $\mathbf{r} \leftarrow \mathbf{r} - \omega\mathbf{t}$; $\nu_r \leftarrow \mathbf{r}^T\mathbf{r}$; (9); $\nu_r \leftarrow \sqrt{\nu_r}$ ▷ (9): MPI global sum ν_r
21: **if** $\nu_r / \nu_b \le \varepsilon_{\mathrm{cclose}}$ **then stop;**
22: $\rho_0 \leftarrow \rho$;
23: **end**

Fig. B2. TPKS algorithm for solving groundwater transport corresponding to the additive Schwarz preconditioned biconjugate gradient stabilized algorithm. The symbol ← denotes that the left-hand side is assigned to the value of the right-hand side, according to Smith et al. (1996). "maxinner" is the maximum of inner iterations; for further notation see Fig. 2.10 of Barrett et al, (1995). The numbers (.) denote the MPI communication points.

multiplication, in order to account for the coupling coefficients $\mathbf{A}_{i,j}$, $i \neq j$ near the subdomain interfaces, b) global collective MPI communication to determine global sums for inner products and global maxima for stopping criteria.

References

Abdelaziz, R., Le, H.H., 2014. MT3DMSP - A parallelized version of the MT3DMS code. J. African Earth Sci. 100, 1–6. doi: 10.1016/j.jafrearsci.2014.06.006.

Bakker, M., Post, V., Langevin, C.D., Hughes, J.D., White, J.T., Starn, J.J., Fienen, M.N., 2016. Scripting MODFLOW Model Development Using Python and FloPy. Groundwater 54, 733–739. https://doi.org/10.1111/gwat.12413.

Balay, S., Brown, J., Buschelman, K., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B., Zhang, H., 2014. PETSc Users Manual Revision 3.4. Work. https://doi.org/10.2172/1178104

Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J.M., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., S., G.W., van der Vorst, H., 1995. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Math. Comput. 64, 1349. https://doi.org/10.2307/2153507.

Bear, J., 1979. Hydraulics of Groundwater. Dover Publications, inc., Mineola, New York.

Berger, M., H. Bokhari, S., 1987. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. IEEE Trans. Comput. https://doi.org/10.1109/TC.1987.1676942.

Böhme, D., 2013. Characterizing load and communication imbalance in parallel applications. Ph.D. dissertation, RWTH Aachen University.

Boman, E.G., Catalyurek, U.V, Chevalier, C., Devine, K.D., 2012. The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering, and Coloring. Sci. Program. 20, 129–150.

Cheng, T., Mo, Z., Shao, J., 2014. Accelerating Groundwater Flow Simulation in MODFLOW Using JASMIN-Based Parallel Computing. Groundwater 52, 194–205. https://doi.org/10.1111/gwat.12047

Delsman, J.R., Hu-a-ng, K.R.M., Vos, P.C., de Louw, P.G.B., Oude Essink, G.H.P., Stuyfzand, P.J., Bierkens, M.F.P., 2014. Paleo-modeling of coastal saltwater intrusion during the Holocene: an application to the Netherlands. Hydrol. Earth Syst. Sci. 18, 3891–3905. https://doi.org/10.5194/hess-18-3891-2014.

Deltares, 2020. iMOD 5.2: *IMOD-WQ (Water Quality)* [WWW Document]. URL https://oss.deltares.nl/web/imod

Diersch, H.-J.G., 2002. FEFLOW reference manual. Inst. Water Resour. Plan. Syst. Res. Ltd 278.

Dolean, V., Jolivet, P., Nataf, F., 2015. An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation. Soc. Industr. Appl. Math. https://doi.org/10.1137/1.9781611974065.

Dong, Y., Li, G., 2009. A parallel pcg solver for MODFLOW. Ground Water 47, 845–850. https://doi.org/10.1111/j.1745-6584.2009.00598.x.

Eijkhout, V., Chow, E., van de Geijn, R., 2015. Introduction to High Performance Scientific Computing, 2nd. lulu.com, p. 553.

Elmroth, E., Ding, C., Wu, Y.S., 2001. High performance computations for large scale simulations of subsurface multiphase fluid and heat flow. J. Supercomput. 18, 235–258. https://doi.org/10.1023/A:1008117130225.

Faneca Sanchez, M., Gunnink, J.L., Van Baaren, E.S., Oude Essink, G.H.P., Siemon, B., Auken, E., Elderhorst, W., De Louw, P.G.B., 2012. Modelling climate change effects on a dutch coastal groundwater system using airborne electromagnetic measurements. Hydrol. Earth Syst. Sci. 16, 4499–4516. https://doi.org/10.5194/hess-16-4499-2012.

Ferguson, G., Gleeson, T., 2012. Vulnerability of coastal aquifers to groundwater use and climate change. Nat. Clim. Chang. 2, 342–345. https://doi.org/10.1038/nclimate1413.

Forum, M.P., 1994. MPI: A Message-Passing Interface Standard. University of Tennessee, Knoxville, TN, USA.

Fox, G.C., 1988. *A Graphical Approach to Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube BT - Numerical Algorithms for Modern Parallel Computer Architectures*, in: Schultz, M. (Ed.), Springer US, New York, NY, pp. 37–61.

Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B., 2010. The scalasca performance toolset architecture. Concurr. Comput. Pract. Exp. 22, 702–719. https://doi.org/10.1002/cpe.1556.

Guo, W., Langevin, C., 2002. User's Guide to SEAWAT: A Computer Program for Simulation of Three-Dimensional Variable-Density Ground-Water Flow. Tech. Water-Resources Investig. B. 6 Chapter A7 01–434.

Hammond, G.E., Lichtner, P.C., Mills, R.T., 2014. Evaluating the performance of parallel subsurface simulators: An illustrative example with PFLOTRAN. Water Resour. Res. 50, 208–228. https://doi.org/10.1002/2012WR013483.

Harbaugh, A.W., Banta, E.R., Hill, M.C., McDonald, M.G., 2000. MODFLOW-2000, The U.S. Geological Survey modular ground-water model — User guide to modularization concepts and the ground-water flow process. U.S. Geological Survey, Open file report 00-92. doi: 10.3133/ofr200092.

Harbo, M.S., Pedersen, J., Johnsen, R., Petersen, K., 2011. Groundwater in a future climate The CLIWAT Handbook.

Henry, H.R., 1964. Effects of dispersion on salt encroachment in coastal aquifers, in "Seawater in Coastal Aquifers. U.S. Geol. Surv. Water- Supply Pap. 1613–C, C70–C84.

Hill, M.C., 1990. Preconditioned Conjugate-Gradient 2 (PCG2), a Computer Program for Solving Ground-water Flow Equations, Water-resources investigations report. Department of the Interior, U.S. Geological Survey.

Huang, L., Wang, L., Shao, J., Liu, X., Hao, Q., Xing, L., Zheng, L., Xiao, Y., 2018. Parallel processing transport model MT3DMS by using openMP. Int. J. Environ. Res. Public Health 15. https://doi.org/10.3390/ijerph15061063.

Hughes, J.D., White, J.T., 2013. Use of General Purpose Graphics Processing Units with MODFLOW. Groundwater 51, 833–846. https://doi.org/10.1111/gwat.12004.

Huizer, S., Oude Essink, G.H.P., Bierkens, M.F.P., 2016. Fresh groundwater resources in a large sand replenishment. Hydrol. Earth Syst. Sci. 20, 3149–3166. https://doi.org/10.5194/hess-20-3149-2016.

Hwang, H.T., Park, Y.J., Sudicky, E.A., Forsyth, P.A., 2014. A parallel computational framework to solve flow and transport in integrated surface-subsurface hydrologic systems. Environ. Model. Softw. 61, 39–58. https://doi.org/10.1016/j.envsoft.2014.06.024.

Ji, X., Li, D., Cheng, T., Wang, X.S., Wang, Q., 2014. Parallelization of MODFLOW using a GPU library. Groundwater 52, 618–623. https://doi.org/10.1111/gwat.12104.

Jung, Y., Pau, G.S.H., Finsterle, S., Doughty, C.A., 2018. TOUGH3 User's Guide, Version 1.0. https://doi.org/10.2172/1461175.

Lang, S., Wittum, G., 2005. Large-scale density-driven flow simulations using parallel unstructured Grid adaptation and local multigrid methods. Concurr. Comput. Pract. Exp. 17, 1415–1440. https://doi.org/10.1002/cpe.900.

Langevin, C.D., Guo, W., 2006. MODFLOW/MT3DMS-based simulation of variable-density ground water flow and transport. Ground Water 44, 339–351. https://doi.org/10.1111/j.1745-6584.2005.00156.x.

Langevin, C.D., Thorne Jr., D.T., Dausman, A.M., Sukop, M.C., Guo, W., 2008. SEAWAT Version 4: A Computer Program for Simulation of Multi-Species Solute and Heat Transport. Tech. Methods https://doi.org/10.3133/tm6A22.

Leonard, B.P., 1988. Universal limiter for transient interpolation modeling of the advective transport equations: The ULTIMATE conservative difference scheme. Nasa, pp. 1–115.

Minderhoud, P.S.J., Erkens, G., Pham, V.H., Bui, V.T., Erban, L., Kooi, H., Stouthamer, E., 2017. Impacts of 25 years of groundwater extraction on subsidence in the Mekong delta. Vietnam. Environ. Res. Lett. 12. https://doi.org/10.1088/1748-9326/aa7146.

Mo, Z., Zhang, A., Cao, X., Liu, Q., Xu, X., An, H., Pei, W., Zhu, S., 2010. JASMIN: A parallel software infrastructure for scientific computing. Front. Comput. Sci. China 4, 480–488. https://doi.org/10.1007/s11704-010-0120-5.

Mulder, J.P.M., Tonnon, P.K., 2011. SAND ENGINE " : BACKGROUND AND DESIGN OF A MEGA-NOURISHMENT PILOT IN THE NETHERLANDS. Coast. Eng. Proc. 1. https://doi.org/10.9753/icce.v32.management.35, management.35.

Naff, R.L., 2008. Technique and Application of a Parallel Solver to MODFLOW. In: Proc. MODFLOW More, pp. 19–21.

Neumann, B., Vafeidis, A.T., Zimmermann, J., Nicholls, R.J., 2015. Future coastal population growth and exposure to sea-level rise and coastal flooding - A global assessment. PLoS One 10. https://doi.org/10.1371/journal.pone.0118571.

Oude Essink, G.H.P., 2003. Mathematical models and their application to salt water intrusion problems. Netherlands Inst. Appl. Geosci. 57–77. https://doi.org/10.13140/2.1.1637.4727.

Oude Essink, G.H.P., Boekelman, R.H., 1996. Problems with large-scale modelling of salt water intrusion in 3D. 14th Salt Water Intrusion Meet 16–31.

Oude Essink, G.H.P., Van Baaren, E.S., De Louw, P.G.B., 2010. Effects of climate change on coastal groundwater systems: A modeling study in the Netherlands. Water Resour. Res. 46. https://doi.org/10.1029/2009WR008719.

Pruess, K., Oldenburg, C., Moridis, G., 2011. TOUGH2 user's guide, version 2.1, LBNL-43134 (revised). Lawrence Berkeley Natl. Lab, Berkeley, CA.

Renaud, F.G., Syvitski, J.P.M., Sebesvari, Z., Werners, S.E., Kremer, H., Kuenzer, C., Ramesh, R., Jeuken, A.D., Friedrich, J., 2013. Tipping from the Holocene to the Anthropocene: How threatened are major world deltas? Curr. Opin. Environ. Sustain. 5, 644–654. https://doi.org/10.1016/j.cosust.2013.11.007.

Scheidegger, A.E., 1961. General theory of dispersion in porous media. J. Geophys. Res. 66, 3273–3278. https://doi.org/10.1029/JZ066i010p03273.

Schneider, A., Kröhn, K.P., Püschel, A., 2012. Developing a modelling tool for density-driven flow in complex hydrogeological structures. Comput. Vis. Sci. 15, 163–168. https://doi.org/10.1007/s00791-013-0207-2.

Schreuder, W.A., 2005. Parallel Numerical Solution of Groundwater Flow Problems, Ph.D. dissertation. University of Colorado.

Smith, B.F., Bjørstad, P.E., Gropp, W.D., 1996. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, New York, NY, USA.

Stive, M.J.F., de Schipper, M.A., Luijendijk, A.P., Aarninkhof, S.G.J., van Gelder-Maas, C., van Thiel de Vries, J.S.M., de Vries, S., Henriquez, M., Marx, S., Ranasinghe, R., 2013. A New Alternative to Saving Our Beaches from Sea-Level Rise: The Sand Engine. J. Coast. Res. 290, 1001–1008. https://doi.org/10.2112/jcoastres-d-13-00070.1.

SURFsara, 2014. Description of the Cartesius system [WWW Document]. URL https://userinfo.surfsara.nl/systems/cartesius/description .

Tudor, B.M., Teo, Y.M., See, S., 2011. Understanding off-chip memory contention of parallel programs in multicore systems. Proc. Int. Conf. Parallel Process 602–611. https://doi.org/10.1109/ICPP.2011.59.

Verkaik, J., Hughes, J.D., Sutanudjaja, E.H., 2015. A Hybrid, Parallel Krylov Solver for MODFLOW using Schwarz Domain Decomposition. AGU Fall Meeting Abstracts.

Vermeulen, P.T.M., Roelofsen, F.J., Minnema, B., Burgering, L.M.T., Verkaik, J., Rakotonirina, A.D., 2019. iMOD User Manual.

Waterloo Hydrogeologic, 2021. *Visual MODFLOW Flex [WWW Document*]. URL https://www.waterloohydrogeologic.com/visual-modflow-flex/

Yang, J., Graf, T., Herold, M., Ptak, T., 2013. Modelling the effects of tides and storm surges on coastal aquifers using a coupled surface-subsurface approach. J. Contam. Hydrol. 149, 61–75. https://doi.org/10.1016/j.jconhyd.2013.03.002.

Zheng, C., Wang, P.P., 1999. MT3DMS : A Modular Three-Dimensional Multispecies Transport Model for Simulation of Advection, Dispersion, and Chemical Reactions of Contaminants in Groundwater Systems. US Army Corps Eng. Eng. Res. Dev. Cent. 220.