Contents lists available at ScienceDirect

Environmental Modelling and Software

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

Distributed memory parallel groundwater modeling for the Netherlands Hydrological Instrument

J. Verkaik[a,b,*], J.D. Hughes[c], P.E.V. van Walsum[d], G.H.P. Oude Essink[a,b], H.X. Lin[e,f], M.F. P. Bierkens[b,a]

[a] *Unit Subsurface and Groundwater Systems, Deltares, Utrecht, the Netherlands*
[b] *Department of Physical Geography, Utrecht University, Utrecht, the Netherlands*
[c] *U.S. Geological Survey Integrated Modeling and Prediction Division, Chicago, United States*
[d] *Wageningen Environmental Research, Wageningen, the Netherlands*
[e] *Delft Institute of Applied Mathematics, Delft University of Technology, Delft, the Netherlands*
[f] *Institute of Environmental Sciences, Leiden University, Leiden, the Netherlands*

ARTICLE INFO

Keywords:
Parallel computing
Distributed memory
Netherlands Hydrological Instrument
Groundwater
Numerical modeling
Integrated modeling

ABSTRACT

Worldwide, billions of people rely on fresh groundwater reserves for their domestic, agricultural and industrial water use. Extreme droughts and excessive groundwater pumping put pressure on water authorities in maintaining sustainable water usage. High-resolution integrated models are valuable assets in supporting them. The Netherlands Hydrological Instrument (NHI) provides the Dutch water authorities with open source modeling software and data. However, NHI integrated groundwater models often require long run times and large memory usage, therefore strongly limiting their application. As a solution, we present a distributed memory parallelization, focusing on the National Hydrological Model. Depending on the level of integration, we show that significant speedups can be obtained up to two orders of magnitude. As far as we know, this is the first reported integrated groundwater parallelization of an operational hydrological model used for national-scale integrated water management and policy making. The parallel model code and data are freely available.

1. Introduction

Worldwide groundwater reserves, being of vital importance for more than 7 billion of people for drinking water, agriculture and industry (Wada et al., 2014), are threatened under changing climate conditions and increasing population. Threats, such as extreme droughts and excessive groundwater pumping, are putting strains on national and regional water authorities to come up with adequate long-term plans for investments and adaptive measures leading to a sustainable and robust water management for the decennia to come.

The Netherlands, with a long history in water management (Huisman, 1998), experienced a severe drought in 1976 which led to the development of several nationwide model applications, ranging from nested systems-based models of the complete water system (Pulles and Sprong, 1985), national-scale finite element groundwater models (Kovar et al., 1992) and national-scale analytical element models (De Lange, 1996). A number of high-resolution regional groundwater model applications were developed from 2000-2013 to support groundwater management by water boards and drinking water companies and eventually the model applications covered most of The Netherlands. Under these developments, in 2005, the national and regional water authorities joined forces in unifying their modeling software and data, together with several national research institutes (former Alterra, now Wageningen Environmental Research; former TNO-BGS and WL | Delft Hydraulics, now Deltares; former MNP, now PBL). This initiative led to the release of the consensus-based National Hydrological Instrument in 2013 (NHI; De Lange et al., 2014). For more than a decade, the NHI provides the Dutch water authorities and consultancies with a modeling environment used for answering actual water related questions, where the continuity of management and maintenance along with new developments of the modeling software and data is secured by the NHI consortium.

Driven by a large amount of available data in the Netherlands, e.g., due to the many boreholes (van der Meulen et al., 2013) and the dense surface-water network covering the Netherlands, these models typically have a high spatial ($\leq$ 250 m) and temporal ($\leq$ 1 day) resolution and

* Corresponding author. Unit Subsurface and Groundwater Systems, Deltares, Utrecht, the Netherlands.
E-mail address: jarno.verkaik@deltares.nl (J. Verkaik).

https://doi.org/10.1016/j.envsoft.2021.105092
Accepted 24 May 2021
Available online 8 June 2021

inherently involve many computational cells and timesteps. For the national model application of the NHI, here briefly referred to as the Netherlands Hydrological Model (NHM), computational resources required for running this integrated model as a single thread on a single core are significant: ignoring surface water flow and transport model component, one simulation year takes ∼9 h computing time, ∼45 GB (gigabyte) of RAM and ∼30 GB of disc storage. This severely limits the practical application of the NHM, e.g. in the Dutch National Water Model (Prinsen et al., 2015) for evaluating future (climate change) scenarios with proposed adaptation measures, which could require simulation time scales up to 100 years. This means that, assuming constant computing time during simulation, 100 years of simulation would roughly take 900 computation hours (or 37.5 days) and ∼3 TB (Terabyte) of storage with a single threaded run on a single core. Such long run times are highly undesirable since typically a large number of simulations are required for model calibration and scenario analysis. Furthermore, such long run times require that servers are stable for long periods of time, which in practice is difficult to accomplish.

Distributed memory parallel computing (see e.g. Eijkhout et al., 2015; Rünger and Rauber, 2013) is a method to significantly reduce computational times and memory, typically following a non-uniform memory access architecture (NUMA). In NUMA, the entire computational grid (memory) is first partitioned into multiple subdomains and one (or more) subdomains is assigned (distributed) to a node, each having local main memory (RAM) and one or more multi-core CPUs (processors). Then, the processor cores solve the problem simultaneously while exchanging necessary data between the nodes through a fast interconnection network using the Message Passing Interface (MPI; Forum, 1994). In the remainder we refer to a (MPI) process as the program that uniquely runs on an associated single processor core.

In this paper we focus on distributed memory parallelization of two of the five hydrological model codes that have been combined in the NHI: the model code for saturated groundwater and the model code for soil-vegetation water transfers in the unsaturated zone (SVAT), see De Lange et al. (2014). The reason for doing this is that the groundwater and SVAT model components are most time consuming and memory intensive. For groundwater, we parallelize the model code MODFLOW (Harbaugh, 2005), the most widely used groundwater flow modeling program in the world, developed by the United States Geological Survey (USGS). MODFLOW has a large open source community of users and developers from many governments, consultancies and research institutes. For the SVAT model component, we parallelize the model code MetaSWAP/TRANSOL (Van Walsum and Veldhuizen, 2011). MetaSWAP is a fast Richards' equation emulator that uses a database of steady state soil moisture profiles for soil physical units, and TRANSOL an emulator of the advection-dispersion equation. MetaSWAP is implicitly connected to MODFLOW through memory at the outer (Picard) iteration level. Since parallelization of MetaSWAP/TRANSOL is done in a relatively straightforward way without requiring communication between processors (hence embarrassingly parallel), the focus in this paper is primarily on parallelizing MODFLOW. Parallelization of the NHI surface water hydrological model codes is beyond the scope of this research. Although this paper focusses on parallelization of NHI model codes used in the national-scale coupled NHM, all the presented methods and software can be used as standalone applications and used at other spatial domains.

Our parallel implementation successfully went through the four phases of DTAP: Development, Testing, Acceptance, and Production (https://en.wikipedia.org/wiki/Development,_testing,_acceptance_and_production) and is operational in the National Water Model (Prinsen et al., 2015) running on eight processor cores. As far as we are aware, this is the first time that such integrated groundwater parallelization is applied in such a setting. Furthermore, our open-source software is readily available to a wide range of hydrogeological modelers at regional water authorities and consultancies.

2. Methods

2.1. General NHM parallelization strategy

Here the NHM (De Lange et al., 2014) is defined as five coupled NHI hydrological model components, see Fig. 1: the groundwater (GW) model component, consisting of 7 confined model layers, the soil vegetation atmosphere for the transfer of quantitative water in the unsaturated zone (SVAT) model component, the unsaturated zone salt transport (UZST) model component, the surface water for sub-catchments (SWSC) model component and the surface water for optimized distributing (SWOD) model component. This definition differs from De Lange et al. (2014) in a way that in our study we exclude the surface water flow and transport model component and include the UZST model component. Model component details are summarized in Table 1; see De Lange et al. (2014) for a more comprehensive description and details on the coupling connectors. In this paper, besides the fully-coupled NHM (or FNHM) including all five models components, we also consider the reduced NHM (of RNHM) that only includes the coupled GW and SVAT model components. The reason for doing this, is that GW-SVAT models are commonly used as regional application of the NHI by a large number of Dutch consortia of provinces, water boards, drinking water companies, and municipalities (see e.g. Snepvangers et al., 2007).

A timing experiment for the fully-coupled NHM on the NHI Windows server (see Fig. 2) shows that the GW, SVAT, and UZST model components are most time consuming and account for 52%, 16%, and 26% of the total simulation run time, respectively. Hence, this motivates parallelizing these model components.

For sake of simplifying coding, our parallelization assumes that each vertical column of cells is assigned to the same subdomain, including the coupled GW-SVAT/UZST cells. Hence, our partitioning of the computational grid is in lateral (horizontal) direction only. This seems a valid assumption since in groundwater models, the number of lateral cells is generally much larger than the number of model layers, and therefore, our approach naturally minimizes the subdomain interface surface area and MPI communication. Fig. 1 illustrates the partitioning of NHI for the case of two subdomains, where the left subdomain is assigned to (parent) process p0 and the right subdomain to (worker) process p1. In our parallelization, we always assume that each subdomain is uniquely assigned to a single processor (core), corresponding to a single MPI process. Furthermore, the parent process is always responsible for gathering all necessary data from the worker processes and coupling towards the surface water sub-catchments. Except for the (off-line/file-based) coupling connectors SWSC ↔ SVAT and SWSC ↔ GW, parallelization of the other NHM connectors is done in a straightforward manner. For SWSC → SVAT and SWSC → GW, all processes read and clip the necessary data from the output files of the SWSC model component in parallel, e.g., sub-catchment river stages for the groundwater model component and groundwater sprinkling for the SVAT model component. For SVAT → SWSC and GW → SWSC, each process aggregates all necessary fluxes in parallel for the surface water sub-catchments, e.g., drainage discharge from the groundwater model, and sends them to the parent process that writes the input file for the SWSC model component. Since subdomain boundaries may divide surface water sub-catchments, e.g., see the light blue eastern sub-catchment in Fig. 1, this means that computing and communicating of partial sums is involved.

Parallelization for the SVAT and UZST model components is straightforward, since both models apply a one-dimensional discretization in model layer (vertical) direction and therefore no lateral MPI communication is needed.

2.2. Parallel performance evaluation

A commonly used indicator for measuring parallel performance is speedup, $S_p = T_1/T_p$, where T_1 is the serial run time (using a single

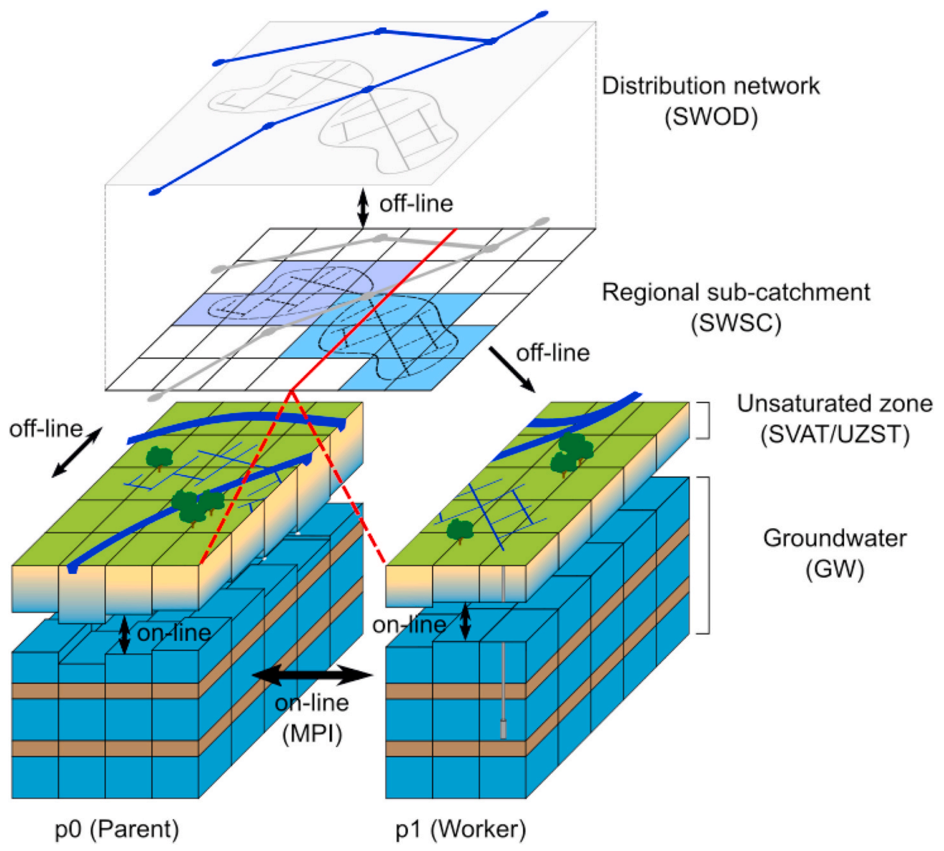

Fig. 1. The fully-coupled NHM covering the water domains considering two parallel processes (p0: Parent, p1: Worker) as an example.

Table 1
Summarized characterizations of the five hydrological model components as part of the NHM (De Lange et al., 2014).

Model Component	Scale of process	Simulation Code	Equation Solved	Spatial dimension	Computation Units	Temporal dimension
GW	Regional	MODFLOW	3D quasi GW flow equation	250 m × 250 m	6,279,002 grid cells	1 day
SVAT	Plot, column	MetaSWAP	1D Richards emulator	250 m × 250 m	550,140 grid cells	1 day
UZST	Plot, column	TRANSOL	1D advection-dispersion equation	250 m × 250 m	550,140 grid cells	10 days
SWSC	Nationwide	MOZART	0D water balance	0.5–5 km^2	8539 polygons	10 days
SWOD	Nationwide	DM	0D water balance	1–25 km	278 nodes	10 days

processor core) and T_p is the parallel run time using p processer cores (see e.g., Eijkhout et al., 2015; Rünger and Rauber, 2013). In practice, evaluating strong scaling means that the problem size (e.g., defined by the number of grid cells) is kept fixed while the number of processor cores is increased. From a modeling point of view, this matches best how users evaluate the parallel performance for their existing models when they have access to multiple processors. In our evaluation, we measure speedup using actual measured wall-clock time instead of CPU time and use the same solver and solver settings for all serial and parallel runs. In our study we have not attempted to determine the optimal performing serial solver or to determine the optimal solver settings. Although for the ideal case $S_{p,\text{ideal}} = p$, in real-world applications this is difficult to obtain. First, from an algorithmic and programming point of view, this depends on the portion of work load that could not or has not been parallelized. If we denote this serial fraction as f, then the well-known Ahmdahl's law (Amdahl, 1967) states that $S_p = (f + (1-f)/p)^{-1}$, and therefore the asymptotic theoretical speedup $S_\infty = 1/f$ when assuming unlimited computer resources. For the fully-coupled NHM model, f is estimated 0.06 which only accounts for the non-parallelized surface water models (Fig. 2 and 6% SWSC + SWOD), thus the maximum theoretical speedup is bounded by $S_\infty = 16.7$. Since other models are also likely to contain serial fractions (e.g., due to solver limitations), we should therefore be realistic about our expectations. Second, assuming that the processors used are connected through a fast interconnection network (low latency and high bandwidth such as a InfiniBand interconnect), achieving parallel performance is typically hampered by communication overhead in the form of wait time (Böhme, 2013). Work load imbalance is defined as $I_p = p\max_{\tilde{p}}(L_{\tilde{p}}/L)$, 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.

2.3. Parallelization for groundwater model component

2.3.1. Relationship to other work

The groundwater model component for the NHM is based on the model code MODFLOW (Harbaugh, 2005), a numerical groundwater flow simulation code using control volumes (cells) for solving the discretized groundwater flow equation. This typically results in (consecutively) solving large and sparse linear systems of equations, accounting for most of the simulation run time. Therefore, parallelization of the MODFLOW linear solver has been subject of considerable research, see Table 2.

Our distributed memory parallelization has similarity with some of the work from Table 2, especially with Schreuder (2005) and Naff (2008). Regarding linear solver preconditioning, we also apply the

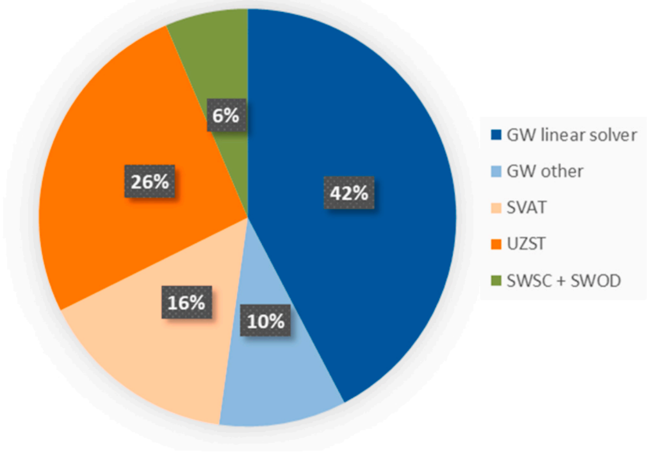

Fig. 2. Fractions of total computing time for the fully-coupled NHM, simulating the year 2006. The total computation took 9 h 17 min on the NHI server. Computing time for the groundwater model component are split into linear solver time ("GW linear solver") and time spent on other routines such as input/output ("GW other").

additive Schwarz preconditioner for solving the linear system in parallel. However, there are notable differences compared to those efforts. First, our approach is fully distributed memory including input and output data. Second, for load balancing models with irregular model boundaries e.g., due to administrative boundaries or geology, we support a robust orthogonal recursive bisection method (Berger and Bokhari, 1987; Boman et al., 2012; Fox, 1988) to divide (partition) the groundwater cells into equally loaded blocks given an arbitrary number of processors. Schreuder (2005) also addressed this problem and developed a partitioning method that iteratively merges cell-weighted blocks while shifting subdomain interfaces. However, this method was concluded not to be robust enough for general purpose. Third, our parallel software only depends on the MPI software library and is therefore relatively easy to compile on multiple platforms. This makes our software accessible to users on a wide range of operating systems, contrasting with other parallelization efforts that use parallel solver libraries primarily developed for the Linux/Unix operating system and have low (to no) support for Windows machines, e.g., the PETSc solver library (Balay et al., 2014). Fourth, our parallel software is open source and actively maintained as part of iMOD (Vermeulen et al., 2019); iMOD is an easy-to-use graphical user interface for the Windows operating system that integrates our accelerated MODFLOW-2005 version with fast subdomain modeling techniques and is extensively used by the NHM user community. Fifth, we add a new modular unstructured parallel solver to MODFLOW-2005, to which we refer to as the Parallel Krylov Solver (PKS; Verkaik et al., 2016, 2015) that is largely based on the UPCG linear solvers (Hughes and White, 2013). Besides assuring minimal dependency on third-party software, another reason for developing the PKS is the ease in reproducing the stopping criteria of the commonly used PCG solver (Hill, 1990) and the flexibility of adding advanced parallel (multi-level) preconditioners in the near future.

2.3.2. *General description*

The main components in our parallelization for groundwater models are a) partitioning of grid cells into subdomains (blocks), b) setting up communication between subdomains, c) reading and writing model input and output files in parallel, and d) parallelization of the linear solver. In this section we highlight the basic concepts of a)-c), referring to Appendix A.1 for more details. For the technical details on solver parallelization d), the reader is referred to Appendix A.2.

2.3.2.1. Subdomain partitioning. In general, parallelization aims to minimize processor idle times, in which a processor does nothing but wait for other processors to finish (see e.g. Rünger and Rauber, 2013). Reduced idle times can be obtained by load balancing and minimizing communication overhead between the processors. Load balancing means that work is equally assigned to the processor cores and in our application directly relates to distributing (partitioning) cells of the computational grid. To minimize communication overhead, we partition the horizontal plane as there is only a few vertical model layers that gives the smallest interface surface area between subdomains (and hence the amount of data communicated). Two (non-overlapping) partitioning methods for partitioning the grid in the horizontal plane are considered: a straightforward method for obtaining equally sized rectangles, here referred to as uniform partitioning, and orthogonal recursive bisection (ORB; Berger and Bokhari, 1987). Fig. 3 illustrates these methods for an example grid, having an irregularly shaped model boundary, considering four partitions (p1 – p4). See Appendix A.1 for details. The red boxes in Fig. 3a show the partitions obtained by straightforward uniform partitioning, clearly showing that the number of active cells vary strongly for each partition. The red boxes in Fig. 3b show the partitions obtained by the ORB partitioning, here using cell weights of value one as illustration. These (user-defined) cells are used within the ORB partitioning to balance the partitions, targeting equal sum of cell weights for each partition. Typically, these cell weights could be chosen to be equal to the number of model layers. It should be noted that our ORB partitioning also includes the Dirichlet boundary conditions (or constant-value active cells; as in example Fig. 3 denoted by index -1). Since these cells are eliminated in the linear solver, this means that a load imbalance may occur in the solving process. Although the ORB partitioning might be optimized for these boundary conditions, this was not a subject for this research. In our approach, the grid is partitioned only once prior to simulation, hence corresponding to static load balancing. Therefore, we neglect the spatio-temporal variation in computing time that might occur during simulation due to changing boundary conditions causing load imbalance. For the NHM this is a reasonable assumption since the number of active cells does not vary in time and the input/output data size and frequency remains constant.

2.3.2.2. Overlap and communication. Discretization of the groundwater flow equation typically results in evaluating a 7-point computation stencil (5-point in the horizontal plane, 3-point in the vertical plane), meaning that the unknown head in a cell implicitly depends on the heads of (at most) six neighboring cells. In a parallel setting, this means that evaluating the discretization of a cell near the subdomain boundary, for instance which is necessary for computing a matrix-vector product within a linear iteration, data from the adjacent subdomain is required and therefore needs to be communicated. To support such local (point-to-point) communication the non-overlapping partitions are expanded to overlapping partitions (see pink boxes for p1 for the example in Fig. 3) by adding one row of cells, so-called halo (or ghost) cells (see the dark greens cells for p1 in Fig. 3). For example, in Fig. 3a with the uniform partitioning processor p1 needs to communicate with p2 and p3, and each processor has exactly two neighbors. However, as can be seen in Fig. 3b, for the ORB partitioning p2 has three neighbors: p1, p3 and p4, and hence there is an additional interface between p2 and p3. This additional communication for ORB is the trade-off for obtaining optimal load balance. For our application this does not seem to be an issue since the amount of data communicated remains low, and local communication overhead is secondary compared to global communication overhead (i.e. communication involving all processors requiring synchronization) and load imbalance. The large benefit of using overlapping partitions is that no additional data (e.g., inter-cell transmissivity) need to be explicitly specified by the user or communicated for evaluating the discretization scheme at the subdomain interfaces. Moreover, this (physical) overlap facilitates the usage of advanced computational schemes with relative ease, such as for applying full-

Table 2
Summary of research done on parallelizing MODFLOW (Cheng et al., 2014; Dong and Li, 2009; Huang et al., 2008; Hughes and White, 2013; Ji et al., 2014; Schreuder, 2005). A distinguish is made between parallelization techniques, software used, linear solver, and measured speedups on specific hardware.

Reference	Shared Memory (OpenMP)	Distributed Memory (MPI)	Graphics processing units (CUDA)	MODFLOW 2000	MODFLOW 2005	Linear Solver	Additional library	Test case	Steady-state (SS) or transient (TR)	Number of Cells (x million)	Hardware	Speedup (threads used)	Remarks
Dong and Li (2009)	✓	–	–	✓	✓	PCG-MIC (Hill, 1990)	–	1: TWRI (Harbaugh et al., 2000); 2: Beishan area refined, north-west China	SS	1: 1 2: 100	Workstation with two 4-core Intel Xeon 2.66 GHz CPUs, 16 GB RAM	1: 1.4 (8) 2:1.3 (8)	Speedup obtained with the slowest gfortran compiled executable
Hughes and White (2013)	✓	–	✓	–	✓	Native UPCG-MILU(0)	–	Hypothetical, 10 layers, heterogeneous, unconfined	SS	10	NVIDIA Tesla C2050	1.6 1.7 (4)	
Ji et al. (2014)	–	–	✓	✓	–	Native PCG-POL	–	Hypothetical,12 layers, homogeneous, confined aquifers	SS TR	33	NVIDIA Tesla C1060	2.5–4	Reorganized the PCG equations
Schreuder (2005)	–	✓[a,c]	–	✓	–	PETSc PCG & additive Schwarz preconditioner	PETSc (Balay et al., 2014)	Rio Grande Decisions Support System model, 5 layers, unconfined, San Luis Valley, Colorado, USA		0.8	64-node/128 processor Intel Xeon 2.4 GHz cluster	26 (48)	
Naff (2008)	–	✓[a,b,c]	–	✓	–	Native PCG & additive/ multiplicative Schwarz preconditioner	–	TWRI (Harbaugh et al., 2000)	SS	1	Heterogeneous cluster 800–3400 MHz	7 (32)	
Huang et al. (2008)	–	✓	–	✓	–	Additive Schwarz coupled at outer iteration level	–	Hypothetical, 2 layers, homogeneous, confined aquifers	SS	0.08	SGI Altix 3700	0.02 (16)	In combination with solute transport simulation using RT3D
Cheng et al. (2014)	–	✓	–	✓	–	Algebraic Multigrid (AMG)	JASMIN	1: Field flow problem at Yanming Lake, China; 2: hypothetical	TR	2: 16	Workstation with four 12-core AMD 2.2 GHz CPUs, 64 GB RAM	1: 6 (40) 2: 22 (32)	AMG gives factor two overhead compared to fastest PCG

[a] Serial input.
[b] Serial matrix assembly.
[c] Serial output.

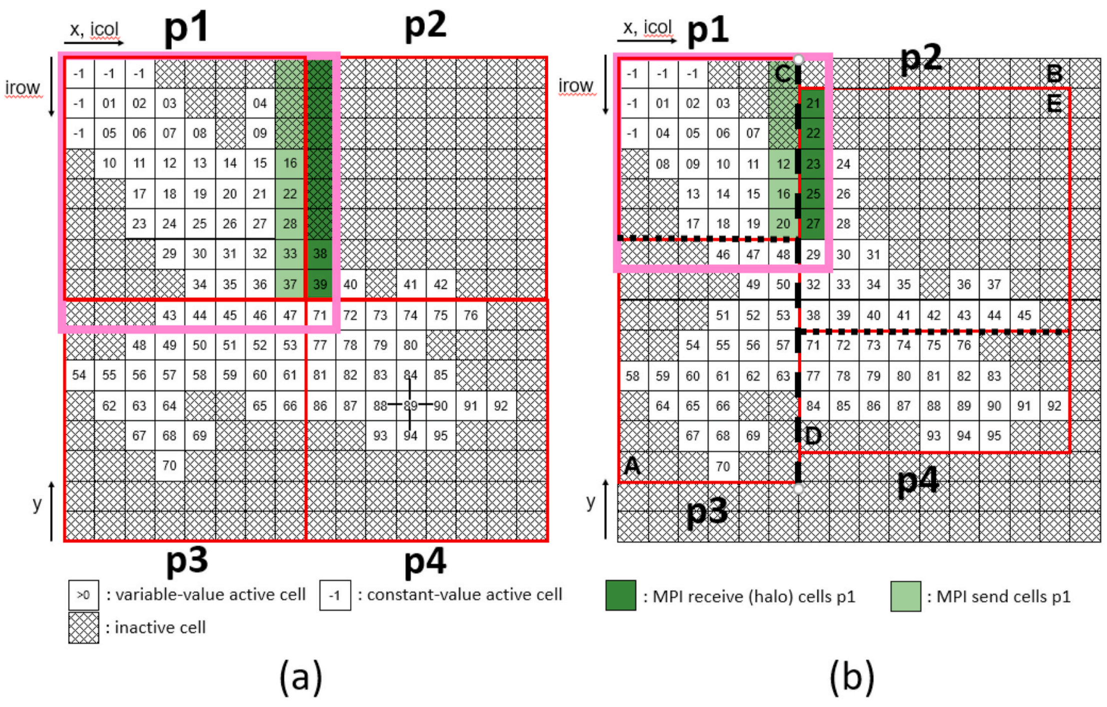
x, icol
irow
y
p1
p2
p3
p4
A
B
C
D
E
>0 : variable-value active cell
-1 : constant-value active cell
: inactive cell
: MPI receive (halo) cells p1
: MPI send cells p1
(a)
(b)

Fig. 3. Example with four processors for uniform partitioning (a) and orthogonal recursive bisection (ORB) partitioning (b), assuming equally weighted cells (both variable-as constant-value/Dirichlet cells) showing two cuts (first: dash black line; second: dotted black line), see Appendix A.1. The red boxes denote the non-overlapping partitions; the pink boxes denote the overlapping partition for processor p1. Green cells denote the communication interface between processors p1 and p2.

tensor anisotropy that requires the evaluation of cross-terms. In our approach, halo cells are treated in a similar way as computational cells regarding input/output and matrix assembly but are different during computation. Each processor is responsible for updating (computing) groundwater heads for its non-overlapping subdomain and halo cells are used to store data received (copies) from adjacent processors. Due to the symmetry of the subdomain overlap, local communication is two-sided, meaning that in addition to receiving data each processor sends data to the neighbor processor (see for the example in Fig. 3 the light green cells).

2.3.2.3. Input and output. Our parallelization supports independent parallel input/output, where each process reads its subdomain data from files that are defined for the entire computational domain and writes its subdomain results to separate files. Parallel input is done from files supporting raster data, point data and line data. For reading raster data, we use the binary geo-referenced iMOD data format (IDF; IMOD-Python Development Team, 2017; Vermeulen et al., 2019) since this file format supports fast unformatted (binary) direct-access read and can be easily visualized with the iMOD graphical user interface. IDF files allow us to efficiently read subdomain data in parallel while keeping memory usage locally. Besides pumping well- and geological fault data, that are read as point and line data respectively, all (static) module and (dynamic) package data (e.g., for rivers and drains) are read from IDF raster files. This means that a significant amount of redundant (no-value) data might be read for sparse raster files e.g., for modeling drainage systems in (semi-)arid areas. Since in the Netherlands the surface water network is dense, the NHM raster data is also dense and therefore the expected redundancy of using IDFs is low. Parallel output is straightforward, where each process writes its separate IDF files or standard MODFLOW ASCII/binary files for its non-overlapping partition. Post-processing these subdomain results might require additional tools, such as iMOD, for merging these data into a single dataset for the total computational grid.

3. Test cases

To evaluate the parallel performance, we consider three test cases with increasing complexity: a hypothetical steady-state regional scale groundwater model; the reduced NHM excluding modeling of salt transport in the unsaturated zone and (dynamic) surface water, simulation for 2006; the fully-coupled NHM with the same initial set-up as the reduced NHM.

3.1. Hypothetical steady-state groundwater model

This hypothetical test case simulates steady-state, regional-scale, groundwater flow in a heterogeneous aquifer for a square computational domain applying uniform partitioning (see Fig. 4, for $12 \times 12 = 144$ subdomains). The model area is 1000 m × 1000 m and two model layers are used (from +10 m to −15 m and −15 m to −30 m, respectively), each having 8000 cells in both x- and y-direction. Hence, the model has 128 million active cells, each having a resolution of 0.125 m × 0.125 m. A hydraulic gradient of 0.01 m/m is specified in the West to East direction, no-flow boundary conditions are specified along the North and South edges of the model, and four equal pumping wells are located in the center of the domain withdrawing a total of 1000 m^3/d from the lowest model layer.

This test case has great similarity with the problem considered by Hughes and White (2013). Our model uses the same simulated multivariate Gaussian ^{10}log(hydraulic conductivity) field for both model layers with an average of 4.81 m/d, a ^{10}log variance of 1.23 and an effective range of 750 m and with the same top and bottom of the aquifer and boundary conditions. Since we have more cells in the x- and y-direction, the hydraulic conductivity of Hughes and White (2013) is downscaled using nearest neighbor interpolation. Our model only considers a single discretization and we assume that the transmissivity of the aquifer is constant (confined option), which makes the problem linear. A linear model gives better insight into the parallel solver performance and is consistent with the assumptions applied in the NHM

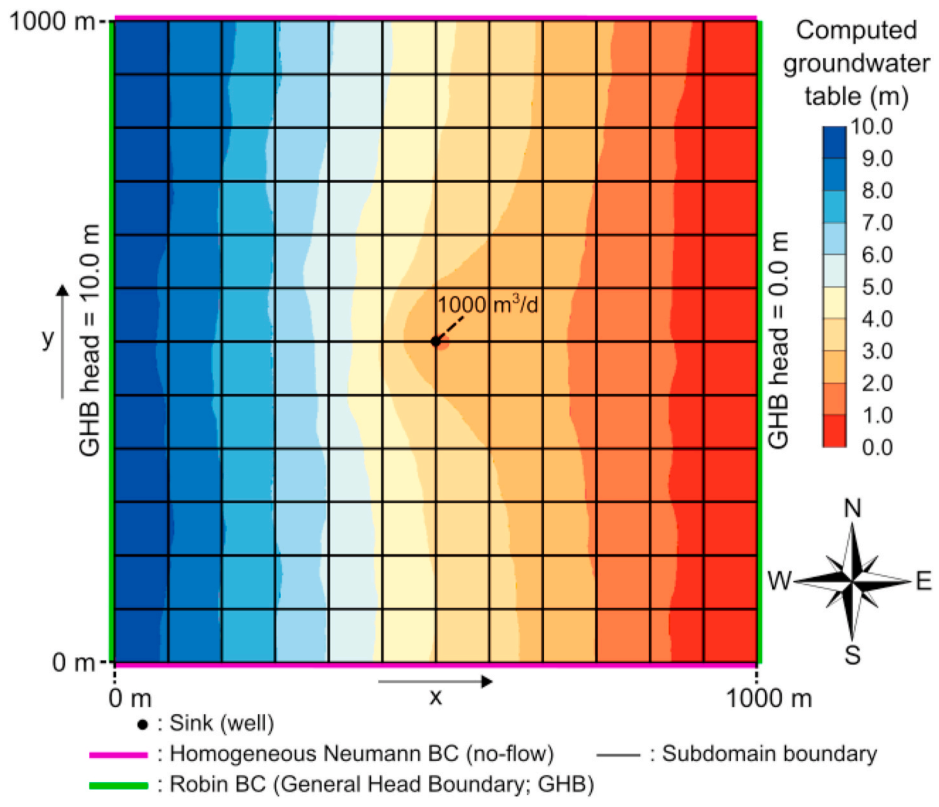

Fig. 4. Hypothetical model domain for an example uniform partitioning of 144 subdomains. Groundwater flow is driven by a head difference of 10 m from East to West, a heterogeneous conductivity, and 4 pumping wells withdrawing a total of 1000 m^3/d. BC: boundary condition.

model.

Convergence with the PKS is obtained for the test case where the stopping criteria are: $\varepsilon_{\text{hclose}} = 0.001$ m (maximum absolute groundwater head difference) and $\varepsilon_{\text{rclose}} = 1.5625 \times 10^{-5}$ m^3/d (~cell size squared times $\varepsilon_{\text{hclose}}$). The simulation starts with initial heads of 0.0 m across the entire domain, see Appendix A.2 for more details. Since the

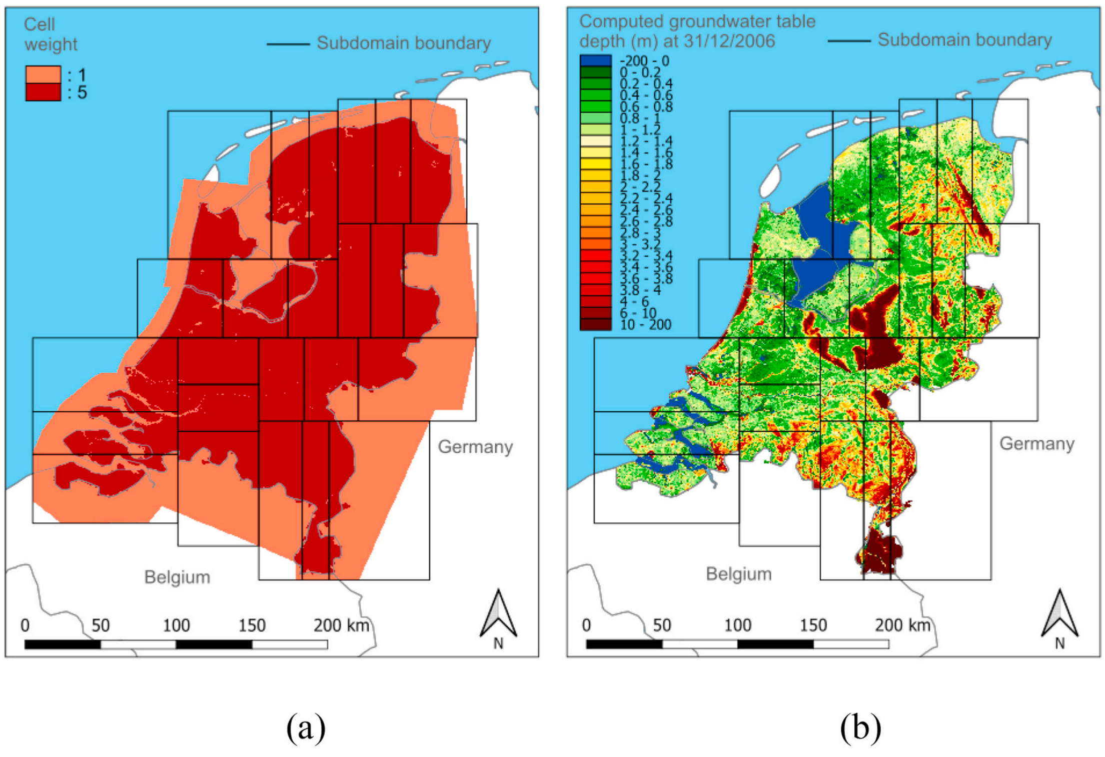

Fig. 5. (A) Orthogonal recursive bisection partitioning cell weights for the NHM and (b) computed groundwater table at 31/12/2006.

domain is square and all cells are active, the uniform partitioning is used and results in optimal load balance, where the number blocks of in x-direction (column direction) is always equal to the number of blocks in y-direction (row direction).

3.2. Reduced and fully-coupled NHM simulating 2006

For both test cases NHM v3.1 is taken (Hoogewoud et al., 2015; see www.nhi.nu). The default groundwater solver settings are used, corresponding to $\varepsilon_{\text{hclose}} = 0.001$ m, $\varepsilon_{\text{rclose}} = 100$ m^3/d, and a maximum of 30 inner iterations for the PKS (see "maxinner" in Fig. 12 of Appendix A.2). The simulation period is 2006. For both the NHM test cases, we apply ORB partitioning since the model boundary is irregular, see Fig. 5 for the example of 24 subdomains. For obtaining a reasonably well load balance for the combined groundwater and SVAT cells, a trial-and-error method is used to obtain the ORB cell weights (see Section 2.3.2). Starting from a uniform cell weight of one for both the groundwater and SVAT cells, the cell weights are obtained by simply increasing the weights for the SVATs cells from 1 to 10. Timing results show that the value of 5 seems to give the overall lowest computing times.

3.3. Hardware and compiler

The parallel performance is evaluated on the Dutch national supercomputer Cartesius (SURFsara, 2014) for the hypothetical test case and the reduced NHM test case, and on the NHI server for the fully-coupled NHM. Cartesius consists of ~1900 computing nodes, running on Linux, that are tightly connected using a fast interconnection network, for a total of ~48 thousand Intel Xeon cores and 128 terabytes of memory. All scaling experiments on the Cartesius are carried out on so-called thin nodes, where each node consists of two Haswell 12-core CPUs (E5-2690 v3) with a total of 64 GB memory. The (single node) NHI server runs on Microsoft Windows and consists of two Intel Haswell 16-core CPUs (E5-2698 v3) with a total of 128 GB memory at the time of evaluating the test cases; the NHI server is a dedicated resource for running NHM simulations that are used for national-scale long term planning. The reason for not evaluating the fully-coupled NHM test case on Cartesius is that the surface water for sub-catchments model code (MOZART) and surface water for the surface water for optimized distributing model code (DM) are not supported to run on Linux. We compiled the coupled MODFLOW-MetaSWAP/TRANSOL model code as part of iMOD v4.0, using the Intel Fortran compiler v15.0 with high level O3-optimization. On Cartesius, the compiled code was linked with the Intel MPI library v5.0 update 3, and for running on the NHI server the code was linked with the MPICH library v1.4.1.

On Cartesius, we use a maximum of four cores per node (hence two cores per CPU) based on trial-and-error testing that has indicated that run-times are shortest using this number of cores. Using a maximum of four cores per node results in 20 idle cores during computation and a relatively low core utilization of 17%. The reason why using more cores in our trial-and-error analyis results in higher run times can likely be explained by large memory requirements for our model applications and the competition of processor cores within a multi-core CPU for the main memory (Tudor et al., 2011). Since we expect this is a hardware related issue inherent to multi-core architectures, this issue is recommended for future research and left outside the scope of this study.

4. Results

4.1. Hypothetical steady-state groundwater model

Fig. 6 shows the measured speedups (a) and total memory usage (b) for the hypothetical test case (see Section 3.1) for our strong scaling experiments on Cartesius up to 144 processor cores. The serial run requires 4 h and 48 min computing time to converge and ~45 GB main memory. The computing time is reduced to 2 min and 40 s using 144 cores (36 nodes), resulting in a speed-up of 108. The absolute groundwater head difference is less than the specified $\varepsilon_{\text{hclose}} = 0.001$ m in each cell in the serial and parallel simulations.

4.2. Reduced and fully-coupled NHM simulating 2006

Fig. 7 shows the measured speedups (a) and total memory usage (b) for the reduced NHM test case on Cartesius and the fully-coupled NHM test case on the NHI server, up to a maximum of 64 and 24 cores, respectively. Maximum speedups of 21.6 and 4.6 are obtained for the NHM test cases, respectively. Besides the ideal (linear) speedup, Fig. 7a also shows the maximum speedup for the fully-coupled NHM according to Amdahl's law when accounting for 6% serial surface water computation (~10 for 24 cores; see Section 2.2).

Regarding accuracy, transient results for the serial and 24-core parallel simulations are evaluated. The root mean squared error values for the entire period of 2006 and considering all model layers are 3.0×10^{-4} m and 1.1×10^{-3} m for the reduced NHM test case and the fully-

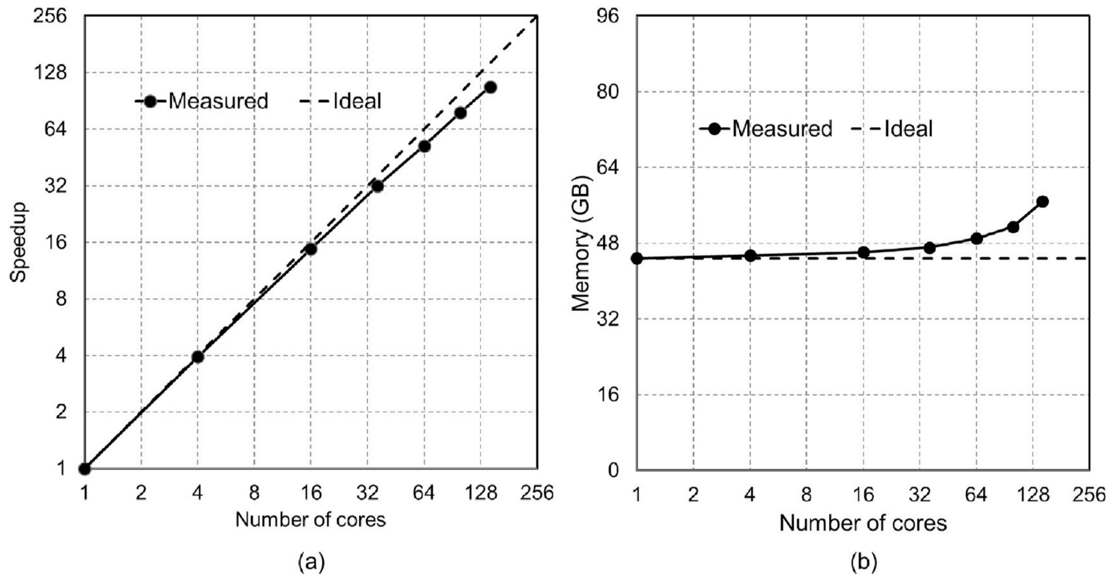

Fig. 6. Measured speedups and total estimated memory usage over all nodes for the hypothetical test case on Cartesius.

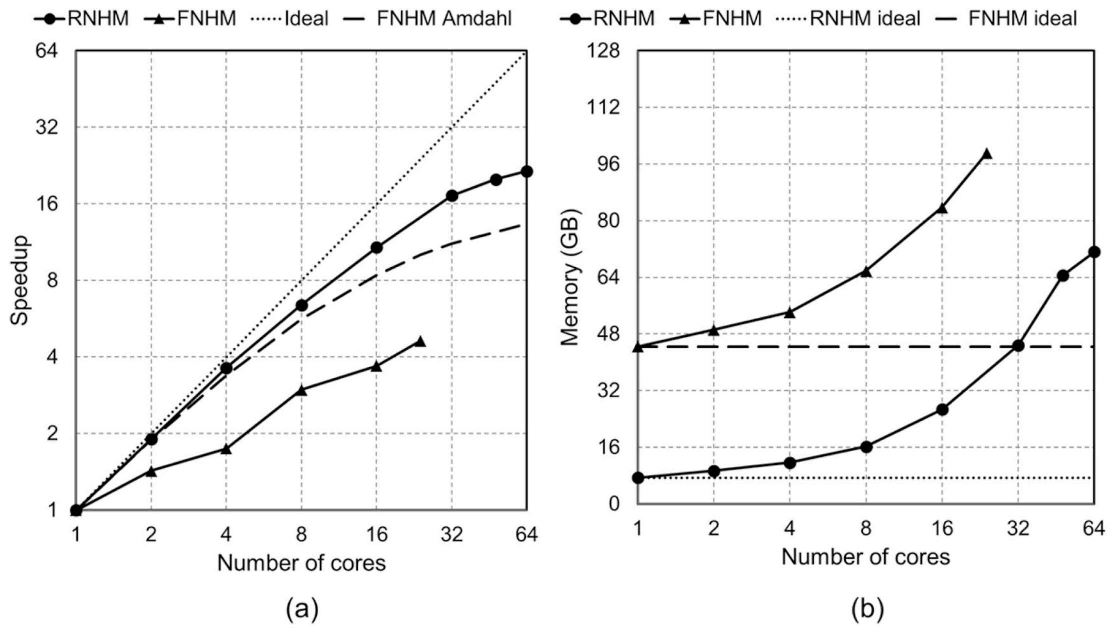

Fig. 7. Measured speedups (a) and total estimated memory usage over all nodes (b) for the reduced NHM (RNHM) test case on Cartesius and the fully-coupled NHM (FNHM) test case on the NHI server. For the fully-coupled NHM, speedups according to Amdahl's law are plotted assuming 6% serial surface water computation. The serial reduced NHM run takes 2 h 23 min 13 s to finish, the serial fully-coupled NHM run 9 h 17 min 16 s.

coupled NHM test case, respectively. For the reduced NHM, the maximum absolute head difference greater than 0.001 m and 0.01 m is exceeded for 3% and 0.1% of the total number of cells during simulation, respectively. For the fully-coupled NHM these values are 7% and 0.4%, respectively.

5. Discussion

Figs. 6a and 7a show that significant speedups are obtained, ranging from two orders of magnitude for groundwater only (MODFLOW; hypothetical test case), one order of magnitude for the reduced NHM excluding dynamic surface water and unsaturated zone transport and less than one order of magnitude for the fully-coupled NHM. However, the speedup curves flatten as the number of cores increases. This can be explained by hardware related issues, non-scalable algorithms/methods, and non-scalable components in implementation.

Regarding hardware, the memory competition issue (see Section 3.3) is likely to contribute to the flattening of the fully-coupled NHM speedups, since a maximum of 12 out of 16 cores per CPU is used compared to 4 out of 12 cores per CPU for the reduced NHM. However, due to the lack of scheduling control options on the NHI Windows server we are not able to quantify this effect.

Concerning non-scalable algorithms, a more important explanation for the flattening of the fully-coupled NHM speedups can be found in the non-parallelized surface water model components which account for ~6% of the total run time. Using Amdahl's law (see section 2.2) and a serial fraction of 6%, a significant flattening of the NHM speedup is expected (see Fig. 7a), where the maximum theoretical speedup using 24 cores is 10 and the measured speedup is 4.6. Furthermore, parallel linear solver iterations might increase with the number of subdomains as a result of low frequency eigen modes that can hamper the linear solver convergence (Dolean et al., 2015; Smith et al., 1996), which can require an additional multi-level preconditioner to improve convergence. For our test cases, however, the maximum observed linear iteration increase is ~15% (see Fig. 8) and suggests that low frequency eigen modes have a relatively limited effect. For that reason, we did not find any need to apply such preconditioner.

With respect to the non-scalable components, the run time behavior is analyzed by cost analysis for the hypothetical test case and the

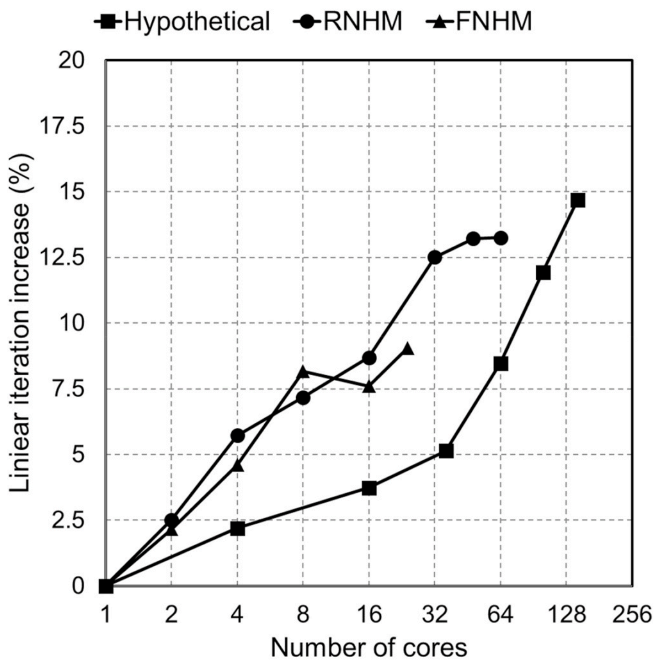

Fig. 8. Linear solver iteration increase for the hypothetical test case (3269 serial iterations), the reduced NHM (RNHM; 18,738 serial iterations) test case, and the fully-coupled NHM (FNHM; 30,390 iterations) test case.

reduced NHM test case on Cartesius using the Scalasca profiling and tracing tool (Geimer et al., 2010). Fig. 9 shows the most significant cost components, where C_p^c denotes the cost of component c for using p cores, defined as the cumulative sum of time spent on c accounting for all processor cores. The total cost C_p of a parallel program is defined as $C_p = pT_p$ and perfect scalability (or cost optimality) is obtained when this cost remains constant for increasing number of cores. Hence, in the ideal case, for each component the relative value $C_p^c/C_1 = C_p^c/T_1$ as shown in Figs. 9a and c should remain constant. The ratio C_p^c/C_p, as shown in

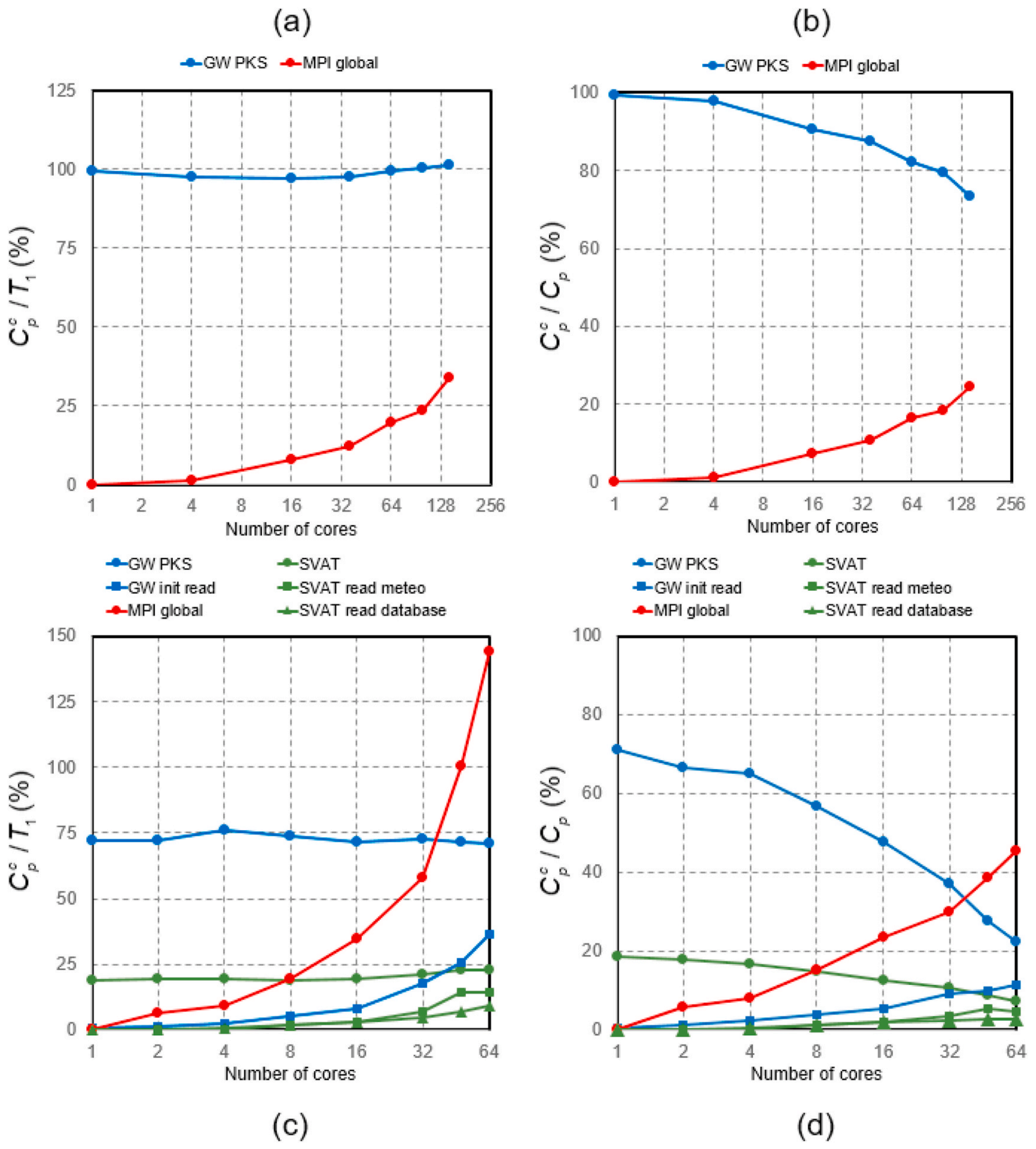

Fig. 9. Scalasca profiling timing results on Cartesius showing parallel component cost C_p^c/T_1 (a: hypothetical test case; c: reduced NHM test case) and relative to parallel cost C_p^c/C_p (b: hypothetical test case; d: reduced NHM test case). In this figure the component "GW PKS" refers to the PKS linear solver computation time; "SVAT" to the computation time; "GW init read" to (redundant) initialization time for reading GW input; "SVAT read database" time for reading the database with soil moisture profiles, "SVAT read meteo" time for reading precipitation and evapotranspiration ASCII grids; and "MPI global" global communication time (load imbalance).

Figs. 9b and d, expresses the contribution of a component to the total parallel run time.

For the hypothetical test case, Fig. 9a ("GWS PKS") shows that majority of the run time is spent in the linear solver and this component has near perfect scalability. However, load imbalance is manifested in global MPI communication wait times ("MPI global" in Fig. 9a). Since the subdomain partitioning is uniform, we suspect that this imbalance is caused by the physical overlap of one row whereas we do not account for this overlap in the load balancing (see Section 2.3.2).

For the reduced NHM test case, scalability of the PKS for the groundwater model component and computations for the SVAT model component are also nearly perfect (see Fig. 9c, "GW PKS" and "SVAT" respectively). However, a strong load imbalance is observed, where ~45% of the run time is spent on waiting when using 64 cores ("MPI global" in Fig. 9d). This load imbalance is very likely related to the groundwater model component and SVAT model components sharing the same partition. For groundwater, active cells exist across the Dutch land-border in model layers 2 to 7, whereas SVAT cells exist only within the Dutch land-border. Using more subdomains enhances this discrepancy near the border and the ORB partitioning becomes less effective. This is illustrated in Fig. 10 for the example with 48 subdomains considering different ORB cell weights. In Fig. 10a, the same cell weights are used as for our NHM test cases (see Fig. 5), where in Fig. 10b cell weights are used that are equal to the number of active groundwater model layers. Tracing analysis shows that subdomains p36 and p42 are responsible for most of the delay, which is mainly caused by global MPI communication. Groundwater cells for subdomains p36 and p42 are not connected to any SVAT cells (Fig. 10b). This results in a significant load imbalance for the SVAT model component, although for groundwater load is well balanced, and a total delay time of ~75% relative to the total

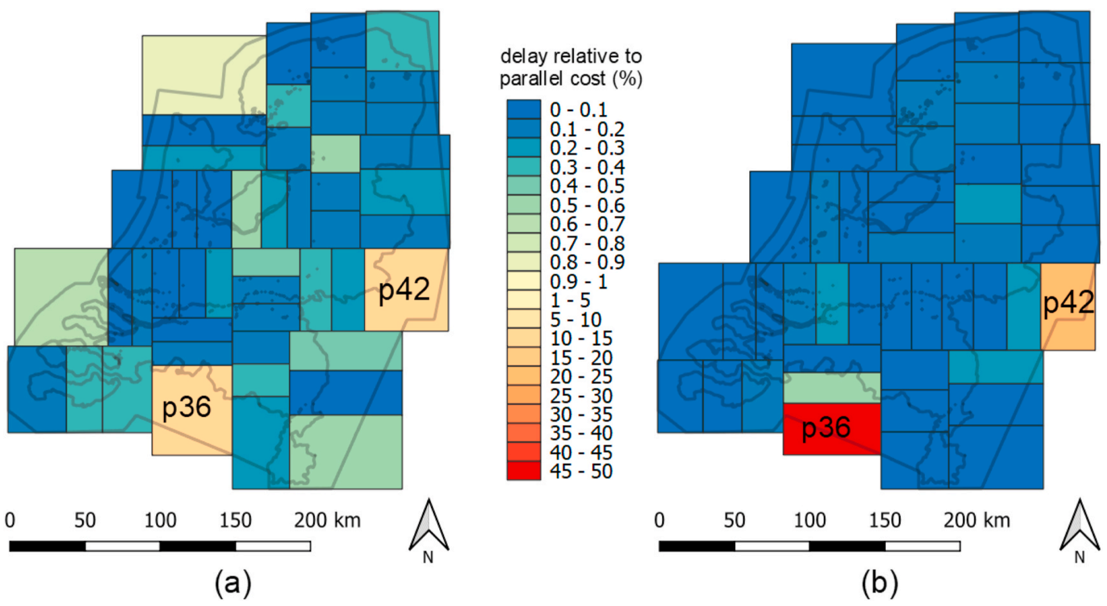

Fig. 10. Delay times for the reduced NHM test case using 48 cores relative to the parallel cost C_p considering (a) the reference ORB cell weights used for our test cases as in Fig. 5 versus (b) ORB cell weights defined as the sum of active groundwater model layers in depth. Process p36 and p42 are most dominant in the delay.

parallel cost. However, as can be seen in Fig. 10a, by using different cell weights subdomains p36 and p42 now have connections to SVATs, and by this we improve the SVAT model component load. On the other hand, we introduce a load imbalance for the groundwater model component. However, this has an overall positive effect on the total delay time that is reduced to ~36% relative to the total parallel cost. This illustrates that ORB partitioning for the NHM is complicated by the coupled SVAT-GW models. A better approach for load balancing might be to decouple the groundwater and SVAT partitions. However, this would require a significant programming effort and is therefore beyond the scope of this current research. Other components that contribute to the flattening of the speedup curved for the reduced NHM (Fig. 7a) are related to input data reading (see Fig. 9c). The component "GW init read" is related to redundant file information required by the operating system. The component "SVAT read meteo" is related to the non-scalable ASCII reading of 1 km precipitation and evaporation input grids, and the component "SVAT read database" related to reading the database with steady states of soil moisture profiles for the (72) soil physical units. Since the SVAT model component is a metamodel that strongly relies on this pre-compiled database (see Van Walsum and Veldhuizen, 2011), in a worst case scenario when all soil physical units are entirely heterogeneous, this means that each processor needs to read the entire database and keep all data in memory.

Regarding RAM memory usage, the SVAT database significantly increases memory usage by ~2.5 GB for the NHM test cases (Fig. 7b) for each processor core added. In practice this means that sufficient memory should be available. However, for the reduced NHM test case on Cartesius the 64 GB RAM per node does not put any constraint on the processor core usage. On the other hand, for the fully-coupled NHM test case on the NHI server (Fig. 7b), the memory increase, together with the large memory usage of ~40 GB for the unsaturated zone salt transport model component, limits the core usage to not exceed the maximum of 128 GB RAM. Together with the increase of read time, this would advocate minimizing the number of soil physical units per subdomain as part of future research. On the other hand, the groundwater model seems to satisfy the distributed memory approach as illustrated in the hypothetical test case (Fig. 6b), although a slight memory increase is observed. One reason for this increase is the inaccurate memory measurement on Cartesius, where we might overestimate the total memory usage by simply multiplying the measured peak amount of memory during simulation (MaxRSS; maximum resident set size) with the number of processes. Another reason might be that the physical overlap of partitions introduces a slight memory increase for increasing number of processes.

Regarding accuracy, small absolute groundwater head differences between the serial and 24-core parallel NHM simulations (see Fig. 11 red lines) were observed that are larger than the linear solver head change stopping criterion of $\varepsilon_{\mathrm{hclose}} = 0.001$ m. The largest difference occurred in a small number of cells, with 99.9% and 99.6% of the cells having mean absolute differences between 0.001 m – 0.01 m in the reduced NHM and fully-coupled NHM, respectively. Errors greater than 1 m were observed during summer 2006 are occur in only 5 cells (0.08‰) and 10 cells (0.16‰) in the reduced NHM and fully-coupled NHM, respectively, and might be related to local convergence issues for the GW-SVAT coupling scheme. The root mean squared error (Fig. 11 blue lines) and the mean absolute root error (Fig. 11 green lines) are lower than or equal to 0.001 m, indicating a good match. Errors greater than $\varepsilon_{\mathrm{hclose}}$ might be caused by the parallel preconditioner for the groundwater model component that differs for each core configuration, resulting in different convergence behavior including the couplings. Furthermore, errors greater then $\varepsilon_{\mathrm{hclose}}$ might be the result of rounding errors caused by single-precision accuracy of the model component connectors, explaining why the fully-coupled NHM seems generally slightly less accurate than the reduced NHM model, or by the parallel aggregation of budgets for the surface water sub-catchments that is non-associative regarding floating point arithmetic in the Parent-Worker mechanism (see Section 2.1). Although small differences may occur, they are found to be acceptable regarding the DTAP software development and therefore we do not find the need to do a more extensive accuracy analysis.

6. Conclusions

We have presented the results of an integrated groundwater parallelization as part of the NHI, focusing on the NHM application. Significant speedups were obtained, ranging from two orders of magnitude for the (non-integrated) groundwater model considering a hypothetical test case (speedup: ~108 using 144 processor cores), to one order of magnitude for the reduced NHM test case excluding the surface water model components and the unsaturated zone transport model component (speedup: ~22 using 64 processor cores), to less than one order of

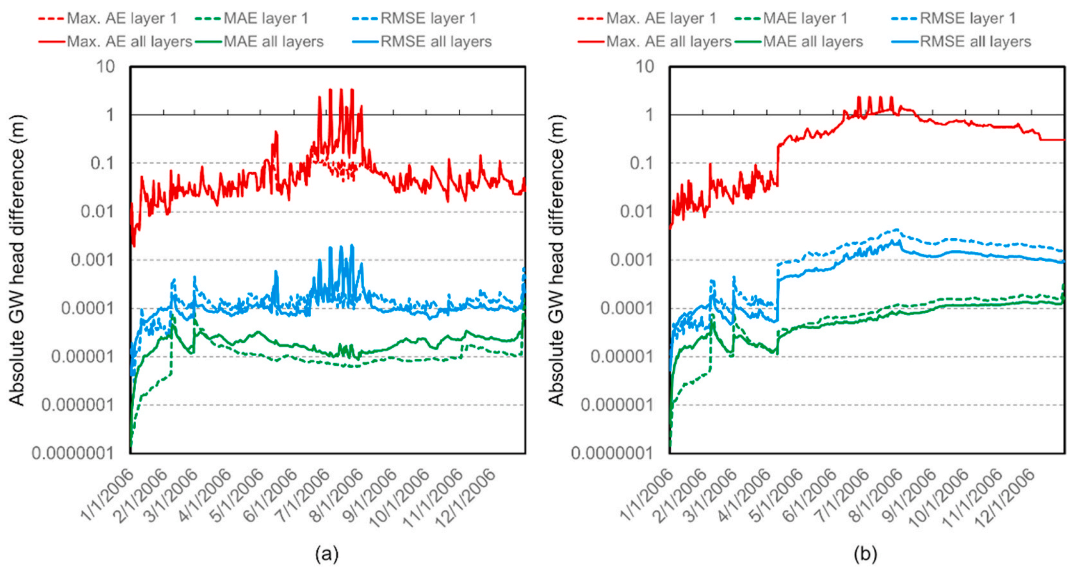

Fig. 11. Absolute groundwater head differences in meters for the reduced NHM test case (a) and the fully-coupled NHM test case (b), where for each test case a parallel run using 24 cores is compared to the reference serial run. Values are plotted for the top model layer and for all seven model layers. Max. AE: maximum absolute error; MAE: mean absolute error; RMSE: root mean squared error.

magnitude for the fully-coupled NHM (speedup: ~5 using 24 processor cores). This clearly shows that coupling more models results in a decrease in speedup, a result that is mostly related to our chosen parallelization strategy. First, we focused on parallelizing the groundwater and SVAT/UZST model components exclusively that are most dominant in computing time and memory usage, therefore ignoring run time due to surface water computations. Second, to parallelize the GW-SVAT/UZST connector in the current model codes with relative ease, we assumed that the groundwater and SVAT/UZST model components share the same partitions resulting in insoluble load imbalance when using many processor cores. Possible improvements of the current parallelization approach would be to parallelize the surface water components and decouple the groundwater and SVAT/UZST partitions to improve load balancing. Furthermore, our analysis showed that parallel data input can be further fine-tuned.

Regarding memory usage, we conclude that our parallelization distributes memory sufficiently for the groundwater model component exclusively but not for the NHM, where the memory might exceed the total available memory. Approximately 2.5 GB RAM per additional processor core is needed for the NHM since the SVAT model component requires that a same large database with soil moisture profiles for spatially varying soil physical units is being read into memory by each process. This suggests the parallelization could be further refined to account partitioning heterogeneity and reduce overall memory usage. Another possibility for reducing memory could be the usage of a RAM disk, where, instead of reading the database in memory at initialization, all processes read the necessary data dynamically from a virtual storage created by local memory.

We conclude that for the NHM, parallel model results are sufficiently accurate, supported by the measured root mean squared errors for a parallel run comparing to the maximum absolute groundwater head change stopping criterion. Differences greater than the stopping criterion were observed, that are likely caused by differences in parallel convergence behavior and rounding errors in the serial and parallel model connectors. However, for most cells these differences are too small to prohibit use of the parallel model application. As a result, we believe our parallelization is suitable for national policy analyses and operational management. As far as we know, this is the first accomplished parallelization and speedup of a large-scale integrated hydrological model using MODFLOW. Our parallelization is open source as part of iMOD and NHI and ready to use for applications that would benefit from reduced computing time and memory usage.

We have shown that parallelizing integrated hydrological model codes is more challenging as the number of model codes increase, each having their own characteristics and model application. As integrated models improve and evolve in time by adding new model components or replacing model components with more sophisticated ones, a redesign of the parallelization approach may be needed. In worst case scenario, this means that many model components proportionally contribute to the run time, inevitably requiring huge parallelization efforts to obtain speedup (Zhang et al., 2020). For the fully-coupled NHM, there are plans to revise the surface water components and to increase the spatial resolution for groundwater. Since run times for the current surface water model components are relatively small, we expect that revising these components will quickly result in surface water run times that are dominant and therefore require parallelization. With the current parallelization strategy, we expect that increasing the groundwater spatial resolution will result in improved speedup when using more cores since the serial fraction of 6% for the surface water model components will then likely be smaller. We also 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 the fully-coupled NHM. However, using more cores efficiently means more memory usage for the SVAT model component and therefore more urgency to reduce memory usage and to equip newest servers with sufficient memory.

Authorship contribution

JV performed the conceptualization, methodology, parallelization of MODFLOW and MetaSWAP/TRANSOL as integral part of NHI and iMOD, pre- and postprocessing of the test cases, simulations, and analysis of the results. JDH wrote the serial code for PKS and assisted on setting up the hypothetical test case. PvW assisted on parallelization of the MetaSWAP/TRANSOL. MFPB, HXL and GHPOE supervised this research and helped with its conceptualization. JV prepared the manuscript with contributions from all authors.

Software availability

The model codes that are used in this paper are mostly open-source and partly in transition to the open domain as part of iMOD (https://oss.deltares.nl/web/imod/home). The NHI model data are freely available at http://www.nhi.nu.

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.

Acknowledgements

We thank Deltares, in special Timo Kroon as project leader of NHI, for making this research possible. We also thank Peter Vermeulen, the iMOD code architect, for his help on incorporating PKS in iMOD. Moreover, we thank Joachim Hunink, Liduin Burgering and Janneke Pouwels for their support in running testing the NHM in parallel on the NHI server. Furthermore, we thank John Donners, Edwin Sutanudjaja, and Martijn Russcher for their support on running jobs on Cartesius. This work was carried out on the Dutch national e-infrastructure with the support of the SURF Cooperative.

APPENDICES.

A Parallelization details

A.1. Subdomain partitioning

To illustrate the grid partitioning, consider the example of Fig. 3 showing an irregular domain consisting of $n_c = 16$ columns and $n_r = 14$ rows, that is partitioned into four partitions ($P = 4$).

Uniform partitioning. Fig. 3a shows an example of a uniform partitioning. The blocks are evenly distributed in the row and column direction, targeting equally shaped rectangles, without accounting for the irregular domain. This method aims at minimizing the edge cuts, hence the number of connections at the interface between the partitions, 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 $P = P_cP_r$, where P_c and P_r are the number of blocks in column and row-direction, respectively, that combination is selected such that P_i equals $P_i = \max\{1, \lfloor n/\sqrt{n_cn_rP^{-1}}\rfloor\}$, where $i = c$ when $n = n_c$ or $i = r$ otherwise. Assuming equal weight of each cell, the load imbalance can be defined as $I = \max\{L_p/N\}$, where L_p is the sum of load for the (non-overlapping) partition p. For this example, processor 1 clearly has the largest number of cells (42), resulting in a load imbalance of $I = 1.68$.

Orthogonal recursive bisection partitioning. The orthogonal recursive bisection recursively bisects intermediate partitions perpendicular to their longest dimension $k \geq 0$ times until $P = 2^k$ partitions each of approximately the same load are obtained. Fig. 3b shows an example of four partitions ($k = 2$), where each active cell (both variable-as constant-value/Dirichlet cells) is assumed to have equal weight. The first intermediate partition is determined by applying a minimum bounding box, enclosing cells A and B. Since the longest dimension (15) is along the columns, a vertical cut (black dashed line) is being made by bisecting the column sum of weights (4, 6, 10, 12, 10, 12, 8, 11, 9, 6, 5, 5, 6, 5, 2, 1), with a total sum of 100, resulting in vertical line between column 6 and 7, such that each new intermediate partition has exactly load 50. Then, two new intermediate partitions are determined: enclosing A and C and enclosing D and E, both having the row direction as the longest dimension, hence rows sum of weights are used to determine the horizontal cuts (dotted black lines). Since in this example all the partitions have the same load (25), the load imbalance is $I = 1$, which is optimal.

A.2. Parallelization of linear solver

Finite volume discretization of the flow equation results in, after (Picard) linearization and eliminating the Dirichlet boundary (constant-value) conditions, in solving the linear system of equations:

$$\mathbf{Ah} = \mathbf{b}, \tag{1}$$

where $\mathbf{h}[\mathrm{L}]$ is the vector of unknown heads, $\mathbf{A}[\mathrm{L}^2\mathrm{T}^{-1}]$ a square, symmetric positive-definite, coefficient matrix with the hydraulic cell-by-cell conductivity, and $\mathbf{b}[\mathrm{L}^3\mathrm{T}^{-1}]$ the vector with groundwater sink/source and storage terms. The corresponding computational stencil is 7-point, hence $\mathbf{A}$ has 7 bands. For solving the linear systems (1) in MODFLOW, we use Krylov subspace acceleration and apply this preconditioner in the preconditioned conjugate gradient (PCG) method (Barrett et al., 1995). Instead of solving (1) directly, the symmetrized preconditioned system

$$\left(\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}\right)\mathbf{M}^{1/2}\mathbf{h} = \mathbf{M}^{-1/2}\mathbf{b}, \quad \mathbf{M}^{-1/2}\mathbf{M}^{-1/2} = \mathbf{M}^{-1} \tag{2}$$

is solved where the matrix $\mathbf{M}$ is called the preconditioner (Barrett et al., 1995; Golub and Van Loan, 1996).

Using block-wise natural node ordering, as illustrated by the positive numbering in Fig. 3, the matrix $\mathbf{A}$ can be written as a block matrix of the form:

$$\begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,P} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ \mathbf{A}_{P,1} & \cdots & \cdots & \mathbf{A}_{P,P} \end{bmatrix}, \tag{3}$$

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. 3a, the block matrix (3) 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 the global matrix (3) is never formed explicitly since each processor only has local coefficients corresponding to a block row.

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), denoted by $\mathbf{M}_{\text{AS}}$:

$$\mathbf{M}_{\text{AS}} \equiv \begin{bmatrix} \mathbf{A}_{1,1} & & & \\ & \mathbf{A}_{2,2} & & \\ & & \ddots & \\ & & & \mathbf{A}_{P,P} \end{bmatrix}, \tag{4}$$

In each PCG iteration, called inner iteration, the preconditioner is being applied once and the system of the form $\mathbf{M}_{\text{AS}}\mathbf{y} = \mathbf{z}$ has to be solved, where $\mathbf{y}$ and $\mathbf{z}$ are denoted as typical search directions. This can be done entirely in parallel: each processor solves the local subdomain problem $\mathbf{A}_{i,i}\mathbf{y}_i = \mathbf{z}_i$ (local solve) in parallel and inaccurately (Brakkee et al., 1998). In our parallelization the local solve is done using an incomplete LU factorization with zero fill in (ILU(0)), similar to PETSc (Balay et al., 2014). Convergence at the $i-$th inner iteration is reached for PCG when the stopping criteria $\|\mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}\|_\infty \le \varepsilon_{\text{hclose}}$ and $\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(i)}\|_\infty \le \varepsilon_{\text{rclose}}$ are satisfied, where the infinity norm is defined as $\|\mathbf{y}\|_\infty \equiv \max_i |y_i|$.

The additive Schwarz preconditioned PCG algorithm in pseudo-code is given by Fig. 12. Parallelization of this method involves a) local MPI point-to-point communication of vectors between subdomains prior to sparse matrix vector multiplication, b) global collective MPI communication to determine global sums for inner products and global maxima for stopping criteria.

```
1:   $\mathbf{x}^{(0)}$ initial guess; $\mathbf{x} \leftarrow \mathbf{x}^{(0)}$; (1);              ▷ (1): MPI local exchange of $\mathbf{x}$
     $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$;
2:   for $i = 1, 2, \ldots, \text{maxinner}$ do
3:       $\mathbf{z} \leftarrow \mathbf{M}_{\text{AS}}^{-1}\mathbf{r}$;                              ▷ Apply additive Schwarz preconditioner
4:       $\rho \leftarrow \mathbf{r}^T\mathbf{z}$; (2);                             ▷ (2): MPI global sum of $\rho$
5:       if $i = 1$
6:           $\mathbf{p} \leftarrow \mathbf{z}$;
7:       else
8:           $\beta \leftarrow \rho / \rho_0$;
9:           $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$;
10:      endif
11:      (3); $\mathbf{q} \leftarrow \mathbf{A}\mathbf{p}$;                           ▷ (3): MPI local exchange of $\mathbf{p}$
12:      $\gamma \leftarrow \mathbf{p}^T\mathbf{q}$; (4); $\alpha \leftarrow \rho / \gamma$;              ▷ (4): MPI global sum of $\gamma$
13:      $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$; $\delta_x \leftarrow \|\alpha\mathbf{p}\|_\infty$
14:      $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{q}$; $\delta_r \leftarrow \|\mathbf{r}\|_\infty$; (5);          ▷ (5): MPI global max of both $\delta_x$ and $\delta_r$
15:      if $\delta_x \le \varepsilon_{\text{hclose}}$ and $\delta_r \le \varepsilon_{\text{rclose}}$ then stop;
16:      $\rho_0 \leftarrow \rho$;
17:  end
```

Fig. 12. Additive Schwarz Preconditioned Conjugate Gradient linear solver algorithm for the Parallel Krylov Solver. 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.

References

Amdahl, G.M., 1967. Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67. Spring, ACM, New York, NY, USA, pp. 483–485. https://doi.org/10.1145/1465482.1465560.

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. https://doi.org/10.2172/1178104. Work.

Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J.M., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H., 1995. Templates for the solution of linear systems: building blocks for iterative methods. S., G.W Math. Comput. 64, 1349. https://doi.org/10.2307/2153507.

Berger, M., Bokhari, S.H., 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.

Brakkee, E., Vuik, C., Wesseling, P., 1998. Domain decomposition for the incompressible Navier–Stokes equations: solving subdomain problems accurately and inaccurately. Int. J. Numer. Methods Fluid. 26, 1217–1237. https://doi.org/10.1002/(SICI)1097-0363(19980615)26:10<1217::AID-FLD693>3.0.CO;2-M.

Cheng, T., Mo, Z., Shao, J., 2014. Accelerating groundwater flow simulation in MODFLOW using JASMIN-based parallel computing. Ground Water 52, 194–205. https://doi.org/10.1111/gwat.12047.

De Lange, W.J., 1996. Groundwater modeling of large domains using analytic elements, Ph.D. dissertation. Delft University of Technology.

De Lange, W.J., Prinsen, G.F., Hoogewoud, J.C., Veldhuizen, A.A., Verkaik, J., Oude Essink, G.H.P., Van Walsum, P.E.V., Delsman, J.R., Hunink, J.C., Massop, H.T.L., Kroon, T., 2014. An operational, multi-scale, multi-model system for consensus-based, integrated water management and policy analysis: The Netherlands Hydrological Instrument. Environ. Model. Software 59, 98–108. https://doi.org/10.1016/j.envsoft.2014.05.009.

Dolean, V., Jolivet, P., Nataf, F., 2015. An introduction to domain decomposition methods: algorithms, theory, and parallel implementation. Society for Industrial and Applied Mathematics. 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.

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

Fox, G.C., 1988. In: Schultz, M. (Ed.), A Graphical Approach to Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube BT - Numerical Algorithms for Modern Parallel Computer Architectures. 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. Concurrency Comput. Pract. Ex. 22, 702–719. https://doi.org/10.1002/cpe.1556.

Golub, G.H., Van Loan, C.F., 1996. Matrix Computations, Third. ed. The Johns Hopkins University Press.

Harbaugh, A.W., 2005. MODFLOW-2005 , the U.S. Geological Survey modular ground-water model — the ground-water flow process. U.S. Geol. Surv. Tech. Methods 253. https://nam03.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdoi.org%2F10.3133%2Ftm6A16&data=04%7C01%7Cn.deep%40elsevier.com%7Ced32d5fbd3884f28957008d93611c144%7C9274ee3f94254109a27f9fb15c10675d%7C0%7C0%7C637600272603421733%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C2000&sdata=Z1ZIvXRLkJ%2BYzT6B40YqZ9FNaCGPVV5CMLCe%2FyJeok8%3D&reserved=0. Geological Survey Techniques and Methods 6-A16.

Harbaugh, A.W., Banta, E.R., Hill, M.C., McDonald, M.G., 2000. User guide to modularization concepts and the Ground-Water Flow Process, MODFLOW-2000 the U.S. Geological Survey modular ground-water model. U.S. Geological Survey Open-File Report 00–92, 121 p. https://pubs.er.usgs.gov/publication/ofr200092.

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.

Hoogewoud, J.C., Van Walsum, P.E.V., de Louw, P.G.B., Hunink, J.C., Prinsen, G.F., Verkaik, J., Veldhuizen, A.A., Kroon, T., van der Bolt, F.J.E., Burgering, L., Groenendijk, P., van der Wal, B., 2015. Veranderingsrapportage LHM 3.1.0: Ontwikkeling, beheer en onderhoud van de landelijke toepassing van het NHI (in Dutch). Utrecht, The Netherlands.

Huang, J., Christ, J.A., Goltz, M.N., 2008. An assembly model for simulation of large-scale ground water flow and transport. Ground Water 46, 882–892. https://doi.org/10.1111/j.1745-6584.2008.00484.x.

Hughes, J.D., White, J.T., 2013. Use of general purpose graphics processing units with MODFLOW. Ground Water 51, 833–846. https://doi.org/10.1111/gwat.12004.

Huisman, P., 1998. Water in the Netherlands, Verslagen en mededelingen - Commissie voor Hydrologisch Onderzoek TNO, vol. 37. https://doi.org/10.1007/978-1-4020-8213-9.

Imod-Python Development Team, 2017. iMOD-Python: make massive MODFLOW models. https://gitlab.com/deltares/imod/imod-python.

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

Kovar, K., Leijense, A., Gan, J.B.S., 1992. Groundwater Model for the Netherlands. Mathematical Model Development and User's Guide. Bilthoven. Report no. 714305002.

Naff, R.L., 2008. Technique and application of a parallel solver to MODFLOW. In: Proceedings of MODFLOW and More, pp. 19–21.

Prinsen, G., Sperna Weiland, F., Ruijgh, E., 2015. The delta model for fresh water policy analysis in The Netherlands. Water resour. OR Manag. 29, 645–661. https://doi.org/10.1007/s11269-014-0880-z.

Pulles, J.W., Sprong, T.A., 1985. Policy Analysis for the Water Management in the Netherlands (PAWN). Rijkswaterstaat Hoofddir. van Waterstaat, Den Haag, Netherlands.

Rünger, G., Rauber, T., 2013. Parallel Programming - for Multicore and Cluster Systems, second ed. Springer.

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.

Snepvangers, J., Minnema, B., Berendrecht, W., Vermeulen, P., Lourens, A., Linden, W., Duijn, M., Bakel, J., Zaadnoordijk, W., Boerefijn, M., Meeuwissen, M., Lagendijk, V., 2007. MIPWA: Water Managers Develop Their Own High-Resolution Groundwater Model Tools.

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.

van der Meulen, M.J., Doornenbal, J.C., Gunnink, J.L., Stafleu, J., Schokker, J., Vernes, R.W., van Geer, F.C., van Gessel, S.F., van Heteren, S., van Leeuwen, R.J.W., Bakker, M.A.J., Bogaard, P.J.F., Busschers, F.S., Griffioen, J., Gruijters, S.H.L.L., Kiden, P., Schroot, B.M., Simmelink, H.J., van Berkel, W.O., van der Krogt, R.A.A., Westerhoff, W.E., van Daalen, T.M., 2013. 3D geology in a 2D country: perspectives for geological surveying in The Netherlands. Netherlands J. Geosci. - Geol. en Mijnb. 92, 217–241. https://doi.org/10.1017/S0016774600000184.

Van Walsum, P.E.V., Veldhuizen, A.A., 2011. Integration of models using shared state variables: implementation in the regional hydrologic modelling system SIMGRO. J. Hydrol 409, 363–370. https://doi.org/10.1016/j.jhydrol.2011.08.036.

Verkaik, J., Hughes, J.D., Sutanudjaja, E., van Walsum, P., 2016. First applications of the new parallel Krylov solver for MODFLOW on a national and global scale. In: AGU Fall Meeting Abstracts.

Verkaik, J., Hughes, J.D., Sutanudjaja, E.H., 2015. A hybrid, parallel Krylov solver for MODFLOW using Schwarz domain decomposition. In: 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.

Wada, Y., Wisser, D., Bierkens, M.F.P., 2014. Global modeling of withdrawal, allocation and consumptive use of surface water and groundwater resources. Earth Syst. Dyn. 5, 15–40. https://doi.org/10.5194/esd-5-15-2014.

Zhang, S., Fu, H., Wu, L., Li, Y., Wang, H., Zeng, Y., Duan, X., Wan, W., Wang, L., Zhuang, Y., Meng, H., Xu, K., Xu, P., Gan, L., Liu, Z., Wu, S., Cheng, Y., Yu, H., Shi, S., Wang, L., Xu, S., Xue, W., Liu, W., Guo, Q., Zhang, J., Zhu, G., Tu, Y., Edwards, J., Baker, A., Yong, J., Yuan, M., Yu, Y., Zhang, Q., Liu, Z., Li, M., Jia, D., Yang, G., Wei, Z., Pan, J., Chang, P., Danabasoglu, G., Yeager, S., Rosenbloom, N., Guo, Y., 2020. Optimizing high-resolution community earth system model on a heterogeneous many-core supercomputing platform (CESM-HR_sw1.0). Geosci. Model Dev. Discuss. (GMDD) 2020, 1–38. https://doi.org/10.5194/gmd-2020-18.