A HYBRID VERSION OF SWAN FOR FAST AND EFFICIENT PRACTICAL WAVE MODELLING[*]

MENNO GENSEBERGER[†] AND JOHN DONNERS[‡]

Abstract. In the Netherlands, for coastal and inland water applications, wave modelling with SWAN has become a main ingredient. However, computational times are relatively high. Therefore we investigated the parallel efficiency of the current MPI and OpenMP versions of SWAN. The MPI version is not that efficient as the OpenMP version within a single node. Therefore, in this paper we propose a hybrid version of SWAN. It combines the efficiency of the current OpenMP version on shared memory with the capability of the current MPI version to distribute memory over nodes. We describe the numerical algorithm. With numerical experiments for important applications we show the potential of this hybrid version, We further optimize the approach and illustrate the behavior for larger number of nodes. Parallel I/O will be subject of future research.

Key words. distributed memory, shared memory, hybrid, MPI, OpenMP, wave modelling

AMS subject classifications. 65Y05, 65Y10, 65Y20, 68Q22, 68Q25, 68W40

1. Introduction. In the Netherlands, for assessments of the primary water defences (for instance [8]), operational forecasting of flooding [7, 10], and water quality studies in coastal areas and shallow lakes (for instance [4]) waves are modelled with the third generation wave simulation software SWAN [1]. However, computational times of SWAN are relatively high. Operational forecasting of flooding and water quality studies require a faster SWAN, at the moment this is a major bottleneck. Assessments of the primary water defences require both a fast and efficient SWAN: in 2011 the assessments needed more than 10000 productions runs, resulting in more than 1500 days of computational time on a single node with 2 Intel quad-core Xeon L5520 processors.

Therefore, extensive benchmarks on different architectures have been performed to investigate the parallel efficiency of the current MPI and OpenMP versions of SWAN on structured computational grids. A small, but typical selection of the results of these benchmarks is reported in this paper (§ 4.1). To understand these results, we studied the numerical algorithm of both the MPI and OpenMP version (§ 3.1 and § 3.2, respectively). The MPI and OpenMP versions do not change the original serial numerical algorithm for parallelization. The numerical algorithm applies a Gauss-Seidel iteration or sweep technique, which may look old-fashioned. However, for this specific application in SWAN, given the processes incorporated and model equations, the approach is a well balanced compromise between broad application range, efficiency, and robustness. Based on the study we propose a hybrid version of SWAN (§ 3.3). This hybrid version combines the efficiency of the current OpenMP version on shared memory with the capability of the current MPI version to distribute memory over nodes. With results for important real life applications we show the potential of this hybrid version (§ 4.2). We further optimize the approach (§ 4.3) and illustrate the behavior for larger number of nodes (§ 4.4). Parallel I/O will be subject of future research.

[†]Deltares, Delft and CWI, Amsterdam, the Netherlands (Menno.Genseberger@deltares.nl).

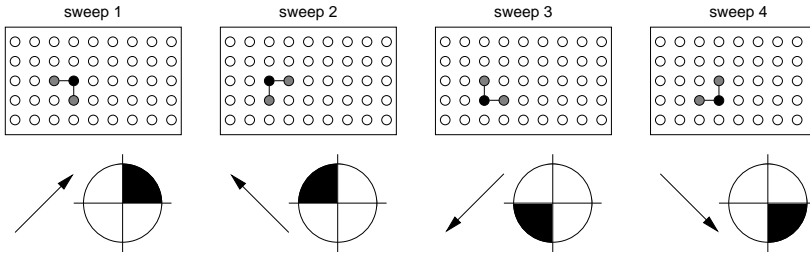[‡]SURFsara, Amsterdam, the Netherlands (john.donners@surfsara.nl).

Fig. 1. *Illustration of the sweep technique for the four quadrants. For every quadrant the arrow indicates the sweep direction and the black bullet represents a computational grid point that is being processed for which information comes from the two grey bullets via the upwind coupling stencil.*
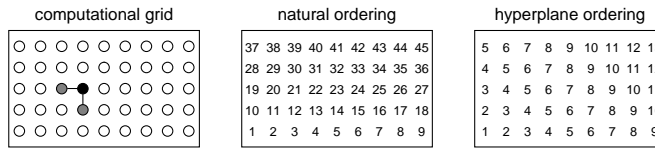

Fig. 2. *Natural ordering and corresponding hyperplane ordering for sweep 1 of the sweep technique. The black bullet represents a point in the computational grid that is being processed for which information comes from the two grey bullets via the upwind coupling stencil.*

2. SWAN. The simulation software package SWAN (Simulating WAves Near-shore) developed at Delft University of Technology [1], computes random, short-crested wind-generated waves in coastal areas and inland water systems. It solves a spectral action balance equation that incorporates spatial propagation, refraction, shoaling, generation, dissipation, and nonlinear wave-wave interactions. The coupling of wave energy via the spectral action balance equation is global over the entire geographical domain of interest. Compared to spectral methods for oceanic scales that can use explicit schemes, SWAN has to rely on implicit upwind schemes to simulate wave propagation for shallow areas in a robust and economic way. This is because typical scales (both spatial, temporal, and spectral) may have large variations when, for instance, waves propagate from deep water towards the surf zone in coastal areas. For spectral and temporal discretization fully implicit techniques are applied. As a consequence the solution procedure of SWAN is computationally intensive. For typical applications these computations dominate other processes like memory access and file I/O.

In the present paper we consider SWAN for structured computational grids (both rectangular and curvilinear) that cover the geographical domain. The spectral space is decomposed into four quadrants. In geographical space a Gauss-Seidel iteration, or sweep technique is applied for each quadrant. This serial numerical algorithm is based on the Strongly Implicit Procedure (SIP) by Stone [13]. Fig. 1 illustrates the sweep technique.

3. Parallel implementation. Given a serial numerical algorithm, in general two parallelization strategies can be followed [3]: type (1): change the algorithm for a high degree of parallelism or type (2): do not change the algorithm but try to implement it in parallel as much as possible. For the serial numerical algorithm of SWAN based on implicit schemes with sweep technique, a strategy of type (2) has an upperbound of maximal parallelism for the computations. For each of the four sweeps
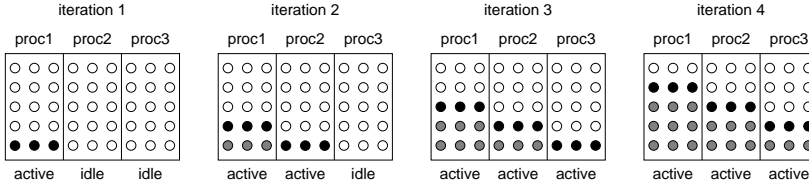
FIG. 3. *Illustration of the block wavefront approach for sweep 1 of the sweep technique. Shown are succeeding iterations in case of three parallel processing units (proc1, proc2, and proc3). The black bullets represent computational grid points that are being updated in the current iteration. The grey bullets were updated in a previous iteration.*

of the sweep technique this upperbound is related to a hyperplane ordering [3, § 4.1]. This depends on the stencil that couples the points in the computational grid: a new value at a point in the computational grid cannot be computed before values are known at neighbouring points that are coupled via this stencil. If computations proceed via some ordering (for instance the natural ordering for sweep 1 as shown in Fig. 2) then the corresponding hyperplane ordering shows those points in the computational grid for which new values can be computed simultaneously (i.e. concurrent computations, in parallel, with opportunity for fine-grained synchronization). These points have the same number in the ordering (a hyperplane), points on which they depend via the coupling stencil have a lower number (data dependency).

3.1. Distributed memory. To reduce computational times of SWAN, Zijlema [15] considered parallelization approaches for distributed memory architectures. The current MPI version of SWAN is based on this work. The approach followed is of type (2): a block wavefront approach for which the author was inspired by a parallelization of an incomplete LU factorization.

In fact, it is based in a more coarse-grained way on the hyperplane ordering for the sweeps of the sweep technique from § 3. For this purpose, the computational grid is decomposed into strips in one direction. The number of computational grid points in this direction is equal or higher than the number of computational grid points in the other direction. Fig. 3 illustrates the block wavefont approach for sweep 1 of the sweep technique (the idea for the other sweeps is similar). In iteration 1, following the dependencies of the upwind stencil, processor 1 updates the values at the computational grid points in the lowest row of strip 1. All other processors are idle in iteration 1. When sweep 1 arrives at the right-most point in the lowest row of strip 1, after the update the corresponding value is communicated to strip 2. Then processor 2 is activated. In iteration 2, processor 1 performs sweep 1 on the next row of strip 1, processor 2 performs sweep 1 on the lowest row of strip 2. Etcetera. Note that not all processors are fully active during start and end phase of this approach. However, for a larger number of computational grid points (compared to the number of processors) this becomes less important. The block wavefront approach is implemented in the current editions of SWAN with MPI [12, 9]. Data is distributed via the decomposition in strips. For each sweep, at the end of every iteration communication between adjacent strips is needed to pass updated values. This global dependency of data may hamper good parallel performance on distributed memory architectures. Note that the MPI version can run on shared memory multi-core architectures too. Furthermore, this approach can be seen as a block (or strip) version of the approach that will be discussed next.
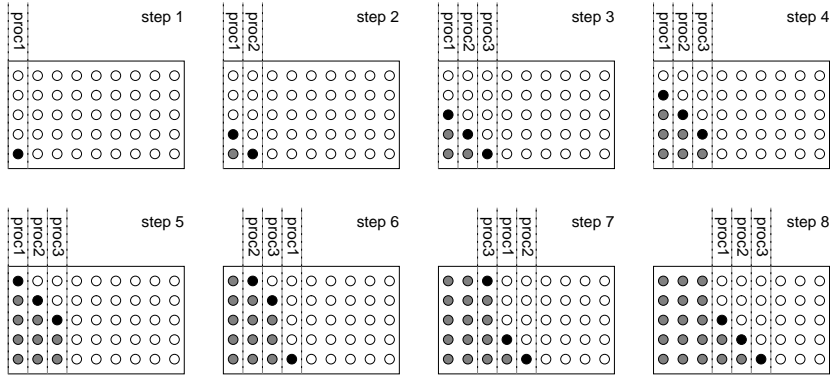
FIG. 4. *Illustration of the pipelined parallel approach based on the hyperplane ordering for sweep 1 of the sweep technique. Shown are succeeding steps in case of three parallel processing units (proc1, proc2, and proc3). The black bullets represent computational grid points that are being updated in the current step. The grey bullets were updated in a previous step.*

FIG. 5. *Illustration of the hybrid approach based on a combination of the block wavefront approach and the pipelined parallel approach for sweep 1 of the sweep technique. Shown are succeeding steps in case of three OpenMP threads (thread 1, thread 2, and thread 3) within two MPI processes (node 1 and node 2). The black bullets represent computational grid points that are being updated in the current step. The grey bullets were updated in a previous step.*
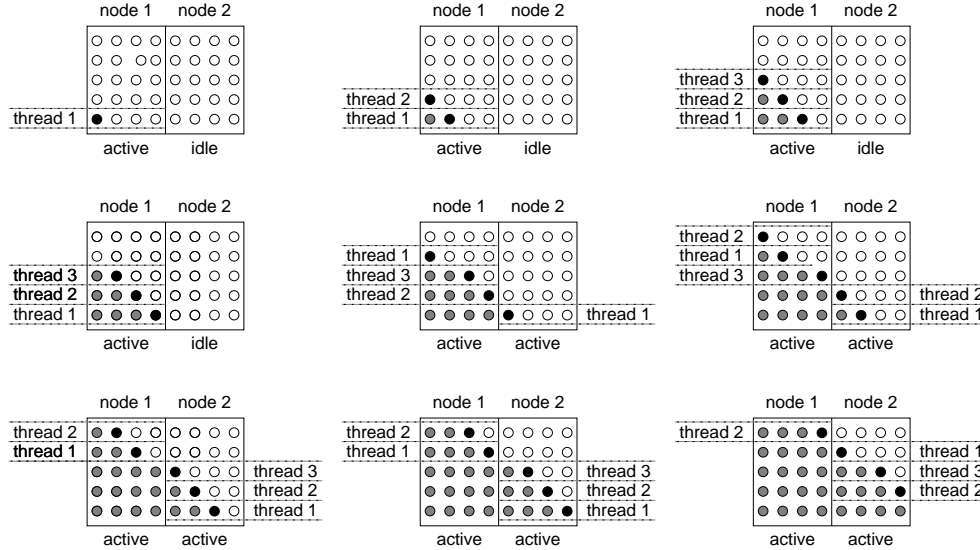
3.2. Shared memory. In [2], Campbell, Cazes, and Rogers considered a parallelization strategy of type (2) for SWAN. The approach is based in a fine-grained way on the hyperplane ordering for the given sweep from § 3. This ordering determines the data dependency and enables concurrent computations with maximal parallelism for type (2). For the implementation with fine-grained synchronization, [2] uses pipelined parallel steps in one direction of the computational grid. Lines with computational grid points in the other direction are assigned to the available processors in a round-robin way. Fig. 4 illustrates the pipelined parallel approach based on the hyperplane

ordering for sweep 1 of the sweep technique (the idea for the other sweeps is similar). In the current editions of SWAN, this approach is implemented on shared memory multi-core architectures (or SM-MIMD, [14, § 2.4, 2012] with OpenMP [5].

3.3. Hybrid version. Further inspection of the approaches used by the MPI and OpenMP versions of SWAN learned us that, conceptually, a combination should be quite straightforward. We illustrated the conceptual approaches for both versions in Fig. 3 and Fig. 4, respectively. Both illustrations were for the same computational grid. Let us reconsider the situation for the approach of the OpenMP version in § 3.2. As the approach is based on the hyperplane ordering for the given sweep, the approach also holds for the transpose of the situation shown in Fig. 4. (In fact [2] uses this transposed situation as illustration.) In this transposed situation lines with computational grid points perpendicular to the other direction are assigned to the available processors in a round-robin way. Now, the point is that this transposed situation for the pipelined parallel approach fits nicely in one strip of the block wavefront approach of § 3.1. The block wavefront approach distributes the strips and for each strip the grid lines are processed efficiently by the pipelined parallel approach within shared memory. In this way the part of the sweeps inside the strips are built up by the pipelined parallel approach and the block wavefront approach couples the sweeps over the strips. Again, all computations can be performed without changing the original serial numerical algorithm. Note that, for one strip the hybrid approach reduces to the pipelined parallel approach, whereas for one processor per strip it reduces to the block wavefront approach.

In Fig. 5 we illustrate this hybrid version for sweep 1 of the sweep technique. To make the link with the actual implementation we give a short description in terms of OpenMP threads and MPI processes. Shown are succeeding steps in case of three OpenMP threads (thread 1, thread 2, and thread 3) within two MPI processes. The black bullets represent computational grid points that are being updated in the current step. The grey bullets were updated in a previous step. Note that, the OpenMP threads on node 1 and node 2 are different threads. With MPI two strips are created: strip 1 is located on node 1, strip 2 on node 2. On node 1, OpenMP starts with the pipelined parallel approach for strip 1. Lines (horizontal for this example) with computational grid points are assigned to the three OpenMP threads in a round-robin way. Node 2 stays idle until sweep 1 arrives at the right-most point in the lowest row of strip 1, after the update the corresponding value is communicated to node 2. Then on node 2 OpenMP starts with the pipelined parallel approach on strip 2. Etcetera. The hybrid version required some subtle modifications in the source code of SWAN for structured computational grids. They are essential to accomodate the combination of OpenMP and MPI:

- the mask array to signal if an OpenMP thread can proceed with processing on the next grid point is now properly defined for multiple subdomains with internal boundaries (and not for only one subdomain with only external boundaries),
- for sending/receiving data to/from neighbouring subdomains: for MPI_SEND and MPI_RECV each task identifier is now uniquely defined for the row on which the data should be exchanged (and not equal to one fixed number) to allow multiple threads to call MPI_SEND and MPI_RECV concurrently without interfering eachother, and
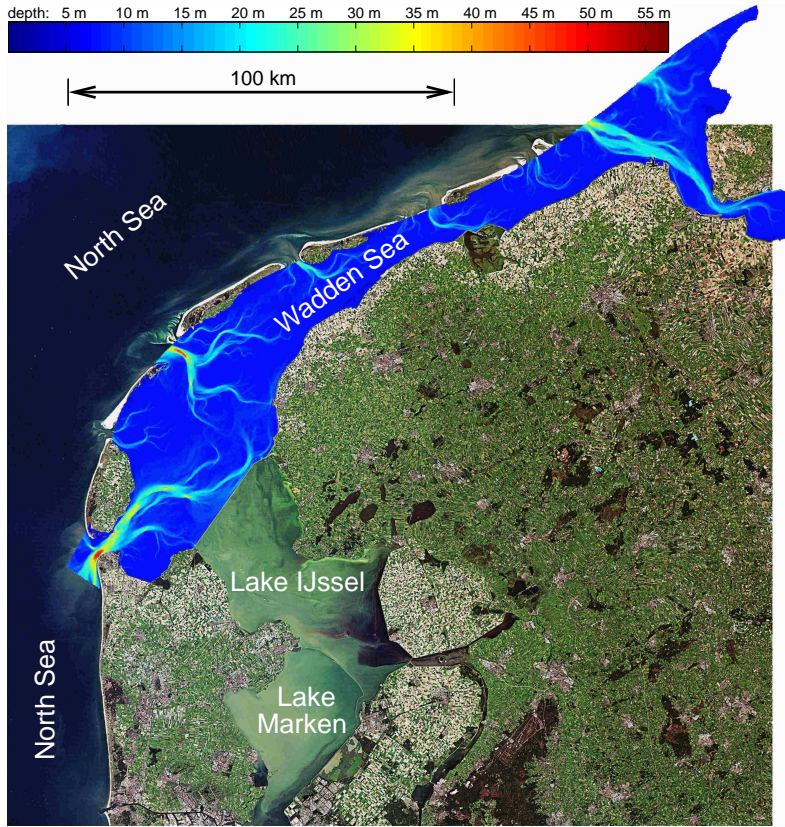- the OpenMP implementation now works in both grid directions (and not only in the first grid direction as in [2]).

Fig. 6. *Bathymetry (mainly blue) of the SWAN model for the Wadden Sea projected on a satellite image of the northern part of the Netherlands (satellite image: ESA). Shown are also lakes IJssel and Marken and a small part of the North Sea.*

4. Benchmarks for important applications. As a central case for the benchmarks a SWAN model is used that has been developed for the assessment of the primary water defences in the northern part of the Netherlands [8]. In 2011 the assessments needed 5400 productions runs with this model, resulting in 1000 days of computational time on a single node with 2 Intel quad-core Xeon L5520 processors (see also § 1). Here the model settings are slightly changed resulting in a wall-clock time of about 1/7 of a single production run of the assessments. The model covers the Dutch part of the Wadden Sea, a complex area of tidal channels and flats sheltered by barrier islands from the North Sea. Fig. 6 shows the bathymetry of this SWAN model projected on a satellite image of the Netherlands. The model is relatively large with a 2280×979 curvilinear computational grid for the geographical domain, resulting in more than 2 million active computational grid points and a required working memory of about 6 GB.

In addition, benchmarks are performed for SWAN models of the North Sea (1318×317 curvilinear computational grid with shortened simulation period and no I/O), Lake IJssel (454×626 rectangular computational grid with full simulation period and I/O), Lake Marken (195×204, 586×614, and 975×1024 curvilinear computational grids with shortened simulation period and no I/O). See Fig. 6 for the locations. The first two models have been developed for operational forecasting of flooding near the

Dutch coast [10] and the Dutch major lakes [7], respectively. The third model has been developed for water quality studies in Lake Marken and is used in combination with a shallow water and advection diffusion solver for modelling resuspention and sedimentation, light penetration, and related ecological effects [4, 6].

Benchmarks are performed on the following hardware:

- Huygens: 16 socket IBM dual 4.7 GHz core Power6 nodes, InfiniBand, IBM PE MPI (IBM pSeries 575 system [14, § 9.2, 2010], SURFsara, the Netherlands),
- Lisa: 2 socket Intel quad-core Xeon L5420 "Harpertown" [14, § 2.8.6, 2008] nodes, 2.50 GHz core, InfiniBand, OpenMPI (SURFsara, the Netherlands),
- WDNZ: 2 socket Intel six-core Xeon E5645 "Westmere-EP" [14, § 2.8.5.2, 2011] nodes, 2.40 GHz core, Gigabit Ethernet, MPICH2 (Deltares, the Netherlands),
- H5: 1 socket Intel quad-core i7-2600 "Sandy Bridge" [14, 2011, § 2.8.5.3 and 2012, § 2.8.4.1] nodes, 3.40 GHz core (hyperthreading: 8 computational threads on 4 cores per node), Gigabit Ethernet, MPICH2 (Deltares, the Netherlands),
- Curie: 2 socket Intel eight-core Xeon E5-2680 "Sandy Bridge-EP" nodes, 2.70 GHz core, InfiniBand, Bull X MPI (Bull B510 bullx system [14, § 3.1.1.1, 2012], CEA, France), and
- Cartesius: 2 socket Intel twelve-core Xeon E5-2690 v3 "Haswell" nodes, 2.60 Ghz core, InfiniBand, Intel MPI (Bull B720 bullx system, SURFsara, the Netherlands).

Benchmarks have been performed for MPI, OpenMP, and hybrid implementations of Deltares[1] SWAN versions 40.72ABCDE (Wadden Sea, North Sea, and Lake IJssel cases) and 40.91AB.8 (Lake Marken case) for Linux 64 bits platforms. Note that for one computational process with one thread, the OpenMP, MPI, and hybrid version are functionally identical to the serial version of SWAN. Standard compiler settings are used as supplied with the Fortran source code at the SWAN website [1]. (Resulting in XL Fortran 12 compiler with level 3 optimization for the Power6 processor and Intel Fortran 11, 12, and 14 compiler with level 2 optimization for the Intel processors.).

During the benchmarks it was checked for all cases that the different combinations (MPI version, OpenMP version, hybrid version, hardware, number of processes / threads) show the same convergence behavior of SWAN. Furthermore, on Cartesius, the Deltares SWAN tesbed was run to verify the validity of the hybrid implementation of Deltares SWAN version 40.91AB.8. This testbed originates from the ONR testbed for SWAN [11]. It runs analytical, laboratory, and field testcases for SWAN for typical functionality and compares results with previous tested versions on different platforms and measured wave characteristics. Based on statistical postprocessing results of the testbed runs can be accumulated in numbers that indicate the model performance on which it can be decided to accept a new SWAN version. For the hybrid implementation of Deltares SWAN version 40.91AB.8 no significant differences were observed.

Timings of the wall-clock time have been performed three times. Results presented here are averages of these timings. To have an indication of the variance (i.e. measurement error), also the average minus the standard deviation and the average plus the standard deviation are included. Shown are double logarithmic plots for wall-clock

---

[1]This Deltares version has some small but subtle additional functionality compared to the official TU Delft version (see website [1]) to enable interaction with a shallow water solver and for wave growth in depth-limited situations like Lake IJssel and Lake Marken.
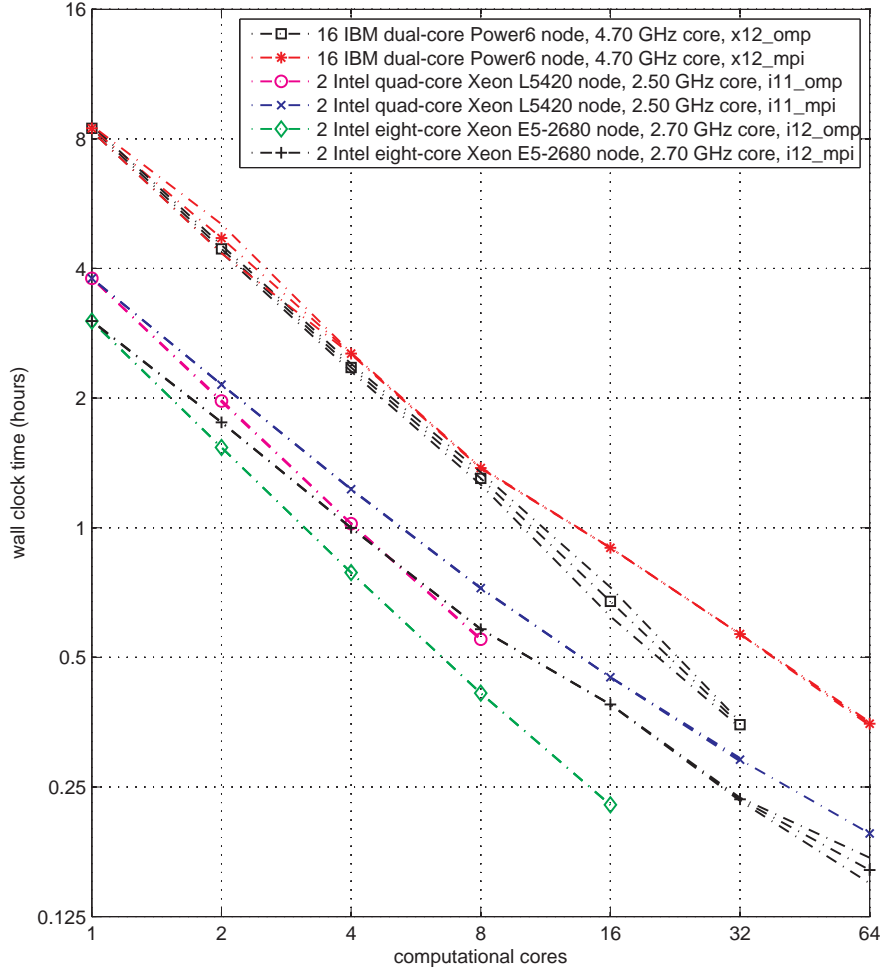
FIG. 7. *Parallel performance of the MPI and OpenMP versions of SWAN for the Wadden Sea case on Curie, Huygens, and Lisa. Abbreviations used: mpi for the MPI version, omp for the OpenMP version, x12 for the XL Fortran 12 compiler, i11 for the Intel 11 compiler, and i12 for the Intel 12 compiler.*

time as a function of the number of computational cores. In case of linear parallel scaling, lines will have a downward slope of 45°.

4.1. MPI version versus OpenMP version. Extensive benchmarks on different architectures have been performed to investigate the parallel efficiency of the current MPI and OpenMP versions of SWAN. A small, but typical selection of the results of these benchmarks is reported here.

Fig. 7 shows benchmark results for both MPI and OpenMP version of SWAN for the Wadden Sea case on the same hardware. It can be observed that already within a single node wall-clock times for the MPI version start to deviate from the OpenMP version (which stays close to the line for linear parallel scaling). This trend goes on for larger numbers of computational cores. For instance, the MPI version on 2 nodes with 2 socket Xeon E5-2680 with in total 32 computational cores has about the same
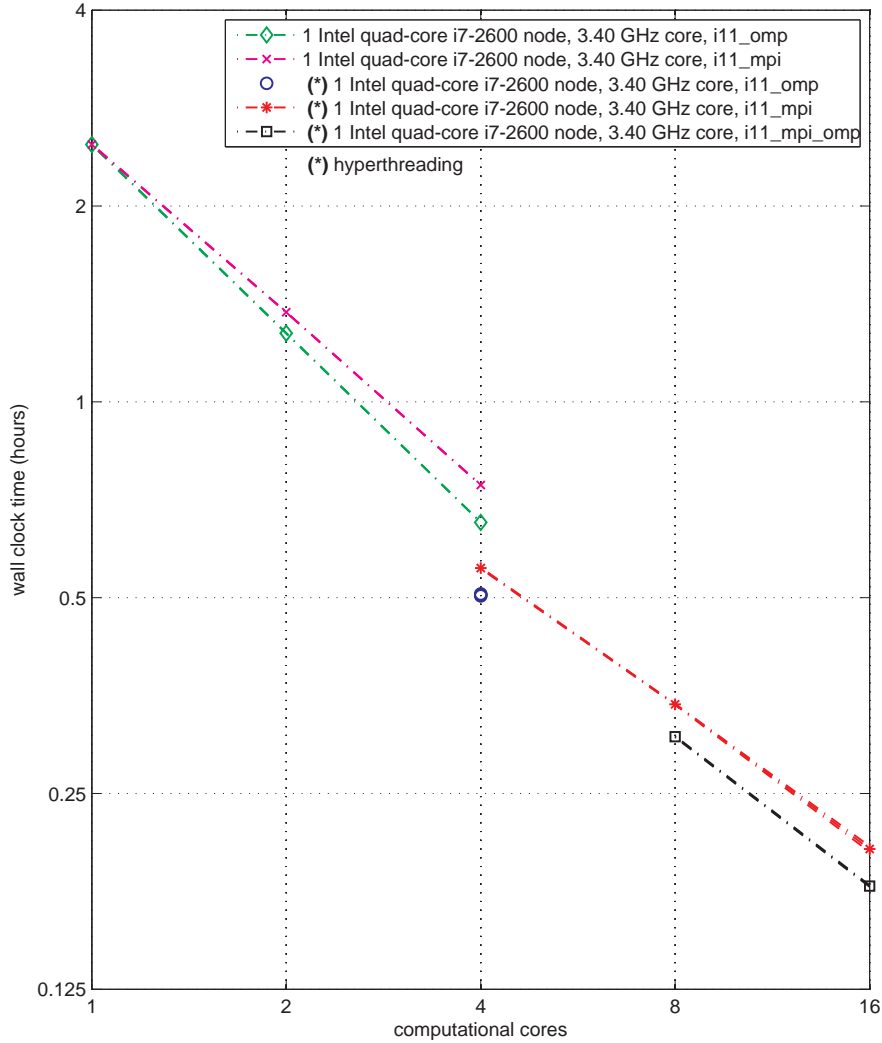
FIG. 8. *Parallel performance of the MPI, OpenMP, and hybrid versions of SWAN for the Wadden Sea case on H5. Abbreviations used: mpi for the MPI version, omp for the OpenMP version, mpi_omp for the hybrid version, and i11 for the Intel 11 compiler.*

wall-clock time as the OpenMP version on 1 node with 2 socket Xeon E5-2680 with 16 computational cores. So, for this hardware the OpenMP version is twice as efficient as the MPI version.

4.2. MPI together with OpenMP: hybrid version. The hybrid version naturally evolves from the MPI and OpenMP versions (§ 3.3). Therefore we expect that it inherits positive performance properties from the OpenMP version as shown in § 4.1. Here, we will start with some numerical experiments to illustrate this.
Fig. 8 shows the parallel performance of the MPI, OpenMP, and hybrid versions of SWAN for the Wadden Sea case on H5. Note that H5 has hyperthreading switched on: in Fig. 8 results with hyperthreading used 8 processes per node with 4 computational cores. Similar to what was observed in § 4.1, we see that the MPI version starts to

deviate from the OpenMP version within a single node. Resulting in a gap of the OpenMP version with the MPI version for 1 node. The hybrid version shows the same gap with the MPI version for 2 and 4 nodes.

In Table 1 we show the corresponding numbers of Fig. 8 for 1 node and 4 nodes. There we also split the wall-clock time in the time needed for the iterations of the solution procedure during and the I/O at the end of the simulation. Time needed for the initialization, before the iterations of the solution procedure start, is several seconds for this Wadden Sea case and is therefore not considered separately. In Table 2 we show results for the same Wadden Sea case, but now for 1 node and 3 nodes of WDNZ. On WDNZ we run 12 processes per node with 12 computational cores. From both tables we conclude that the hybrid version indeed combines the efficiency of the OpenMP version per node with the MPI version to use more than one node. This results in a hybrid version that is faster than the MPI version.

The widening of the gap for more cores per node (see end of § 4.1) can also be observed for the hybrid version: compare Table 1 and Table 2 for the MPI and hybrid versions on multiple nodes. This is of importance as the current trend is that the number of cores per node increases for new hardware. Furthermore, Table 1 and Table 2 indicate that the current implementation of I/O should be further investigated for possible improvements. If we concentrate only on the wall-clock time needed for the iterations then we also see that the gap increases when going to more nodes and/or more cores per node. This is confirmed also for the case of the North Sea on H5 in Table 3.

TABLE 1
Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top) and 4 nodes (bottom) of H5 (4 cores per node and hyperthreading) for the Wadden Sea case.

1 node, 8 processes / threads	MPI version (t1)	OpenMP version (t2)	t1 / t2
wall-clock time full simulation (m)	33.31 ± 0.07	30.28 ± 0.10	1.1000
wall-clock time iterations (m)	26.59 ± 0.03	24.80 ± 0.04	1.0724
wall-clock time I/O at end (m)	6.71 ± 0.11	5.48 ± 0.14	1.2247
4 nodes, 32 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time full simulation (m)	12.32 ± 0.13	10.81 ± 0.03	1.1398
wall-clock time iterations (m)	8.98 ± 0.06	7.79 ± 0.06	1.1527
wall-clock time I/O at end (m)	3.33 ± 0.18	3.01 ± 0.09	1.1066

TABLE 2
Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top) and 3 nodes (bottom) of WDNZ (12 cores per node) for the Wadden Sea case.

1 node, 12 processes / threads	MPI version (t1)	OpenMP version (t2)	t1 / t2
wall-clock time full simulation (m)	33.31 ± 0.04	22.75 ± 0.22	1.4643
wall-clock time iterations (m)	24.35 ± 0.10	18.26 ± 0.06	1.3335
wall-clock time I/O at end (m)	8.96 ± 0.14	4.49 ± 0.28	1.9962
3 nodes, 36 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time full simulation (m)	14.77 ± 0.13	11.27 ± 0.02	1.3110
wall-clock time iterations (m)	9.84 ± 0.12	7.08 ± 0.07	1.3891
wall-clock time I/O at end (m)	4.93 ± 0.25	4.18 ± 0.08	1.1789

4.3. Further optimization and behavior inside a node. Current hardware trends show an increase in the number of computational cores per processor whereas multiple processors share memory inside a node. We think it is important to take notice of this. Therefore, here we first try to further optimize the MPI, OpenMP, and

Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top) and 4 nodes (bottom) of H5 (4 cores per node and hyperthreading) for the North Sea case.

1 node, 8 processes / threads	MPI version (t1)	OpenMP version (t2)	t1 / t2
wall-clock time iterations (m)	62.28 +/- 0.03	52.82 +/- 0.08	1.1790
4 nodes, 32 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time iterations (m)	22.65 +/- 0.05	18.28 +/- 0.06	1.2389

TABLE 4
Wall-clock time in minutes for the iterations of the MPI, OpenMP, and hybrid (2 MPI processes, one MPI process per socket with 12 OpenMP threads each) versions on one Cartesius node using all 24 cores for the Wadden Sea case for different types of pinning the MPI processes / OpenMP threads to the cores.

type of pinning	MPI version	OpenMP version	hybrid version
none	7.874 +/- 0.044	5.405 +/- 0.125	5.339 +/- 0.009
scatter	7.659 +/- 0.008	5.357 +/- 0.026	5.346 +/- 0.008
compact	7.645 +/- 0.013	5.236 +/- 0.026	5.335 +/- 0.009

hybrid version before we scale up to larger number of nodes. We do this for Cartesius (2 processors / sockets per node, 12 cores per processor).

We start with some numerical experiments to investigate the effect of the position of the computational processes on specific locations (cores, processors) inside a node. We consider the iterations of the Wadden Sea case (i.e. no I/O).

Table 4 shows the wall-clock time in minutes of the MPI, OpenMP, and hybrid (2 MPI processes, one MPI process per socket with 12 OpenMP threads each) versions on one Cartesius node using all 24 cores. We investigated different types of pinning the MPI processes / OpenMP threads to the cores:

none: computational processes are placed randomly during execution of SWAN at runtime (i.e. not pinned to cores),

scatter: computational processes of neighbouring strips (MPI) or lines (OpenMP) are placed as far as possible from eachother, and

compact: computational processes of neighbouring strips (MPI) or lines (OpenMP) are placed as close as possible to eachother.

From the table we may conclude that, when using all cores inside a node, the lowest wall-clock times are for compact pinning of the computational processes for both the MPI, OpenMP, and hybrid version.

Fig. 9 shows the parallel performance of the serial, MPI, and OpenMP versions on one Cartesius node as a function of the number of cores used. For the MPI and OpenMP version two ways of pinning the processes / threads to the cores are shown:

compact: computational processes of neighbouring strips (MPI) or lines (OpenMP) are placed as close as possible to eachother and

spread: computational processes of neighbouring strips (MPI) or lines (OpenMP) are placed in corresponding order but spread over the free cores as much as possible.

For example: if only 6 cores are used then for compact the computational processes 1, 2, 3, 4, 5, and 6 are placed on physical cores 0, 1, 2, 3, 4, and 5, respectively. For spread they are placed on physical cores 0, 4, 8, 12, 16, and 20, respectively. From the figure we may conclude that, when using not all cores inside a node, it is beneficial for the wall-clock time to "spread" the computational processes over the cores. Furthermore, in Fig. 9 it can be seen that this effect dissapears when using
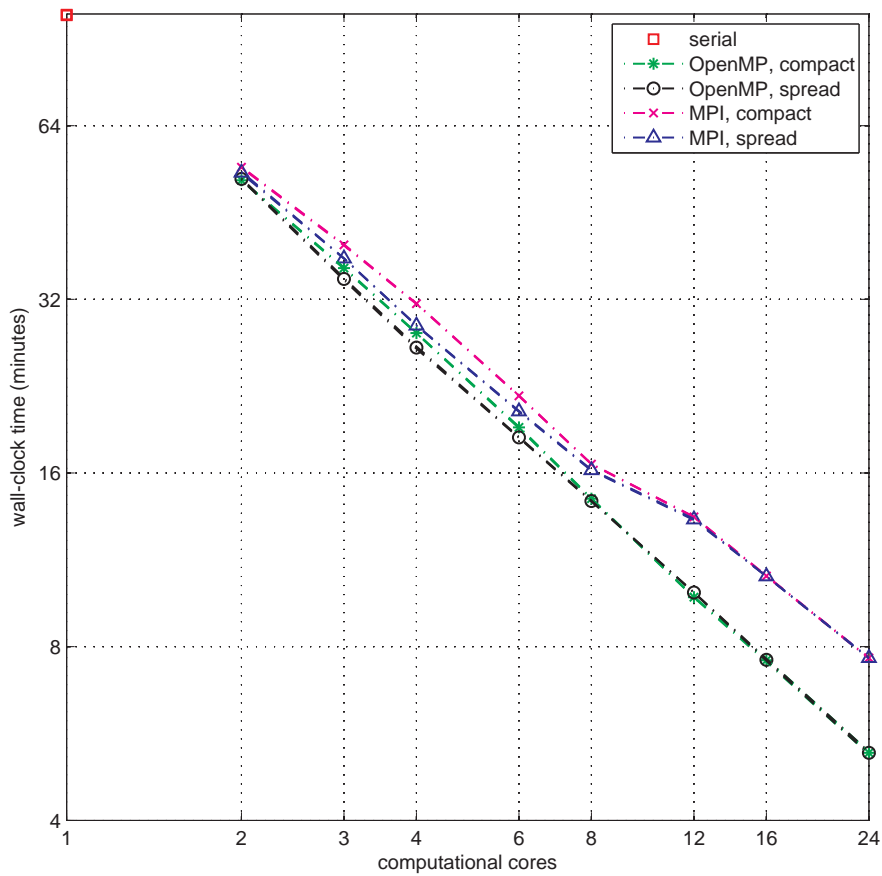
Fig. 9. *Parallel performance of the serial, MPI, and OpenMP versions of SWAN for the Wadden Sea case on one Cartesius node as a function of the number of cores used. For the MPI and OpenMP version two ways of pinning the processes/threads to the cores are shown: compact and spread. See § 4.3 for further explanation. Shown is the wall-clock time in minutes for the iterations.*

more than 8 cores on Cartesius and that then also the difference between MPI and OpenMP becomes more prominent.

A similar observation can be made from Fig. 10. This figure shows the parallel performance of the hybrid version of SWAN on one Cartesius node using all 24 cores as a function of the number of MPI processes. Each MPI process uses (24 / number of MPI processes) OpenMP threads. Note that for one MPI process the hybrid version reduces to the original OpenMP version with 24 OpenMP threads (most left), whereas for 24 MPI processes it reduces to the original MPI version (most right). For each MPI process of the hybrid version the OpenMP threads are pinned compact to the cores, for the next MPI process in the ordering of the algorithm the OpenMP threads are pinned compact to the next cores in the ordering of the node. For the MPI and OpenMP versions the MPI processes respectively OpenMP threads are pinned compact to the cores of the node. Shown is the wall-clock time in minutes for the iterations.

Based on the previous numerical experiments we conclude that, when using all cores of a Cartesius node, optimal settings for the MPI, OpenMP, and hybrid versions is
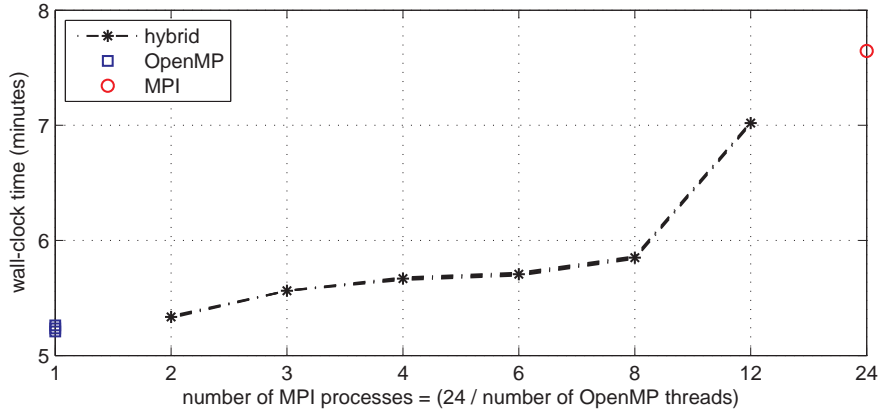
FIG. 10. *Parallel performance of the hybrid version of SWAN for the Wadden Sea case on one Cartesius node using all 24 cores as a function of the number of MPI processes. Each MPI process uses (24 / number of MPI processes) OpenMP threads. See § 4.3 for further explanation. Shown is the wall-clock time in minutes for the iterations.*

pinning the MPI processes and/or OpenMP threads compact to the cores. We will use these settings for the remainder of this paper.

TABLE 5
Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top), 2 nodes (middle), and 4 nodes (bottom) of Cartesius (24 cores per node) for the Wadden Sea case.

1 node, 24 processes / threads	MPI version (t1)	OpenMP version (t2)	t1 / t2
wall-clock time iterations (m)	7.645 ± 0.013	5.236 ± 0.026	1.4601
2 nodes, 48 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time iterations (m)	4.530 ± 0.002	2.994 ± 0.002	1.5130
4 nodes, 96 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time iterations (m)	2.540 ± 0.012	1.668 ± 0.005	1.5228

TABLE 6
Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top), 2 nodes (middle), and 4 nodes (bottom) of Cartesius (24 cores per node) for the Lake IJssel case.

1 node, 24 processes / threads	MPI version (t1)	OpenMP version (t2)	t1 / t2
wall-clock time full simulation (m)	117.229 ± 0.153	60.600 ± 0.350	1.9345
wall-clock time iterations (m)	115.493 ± 0.165	57.736 ± 0.410	2.0004
wall-clock time I/O at end (m)	1.736 ± 0.012	2.863 ± 0.060	0.6064
2 nodes, 48 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time full simulation (m)	65.509 ± 0.051	45.906 ± 0.119	1.4270
wall-clock time iterations (m)	64.373 ± 0.055	43.197 ± 0.151	1.4902
wall-clock time I/O at end (m)	1.136 ± 0.004	2.709 ± 0.032	0.4193
4 nodes, 96 processes / threads	MPI version (t1)	hybrid version (t2)	t1 / t2
wall-clock time full simulation (m)	38.282 ± 0.025	27.480 ± 0.052	1.3931
wall-clock time iterations (m)	37.351 ± 0.052	25.101 ± 1.071	1.4880
wall-clock time I/O at end (m)	0.931 ± 0.027	2.379 ± 1.019	0.3913

With this knowledge / settings we extend the numerical experiments from § 4.2 with results on 1, 2, and 4 Cartesius nodes for the Wadden Sea case (iterations, no I/O) in Table 5 and the Lake IJssel case (full simulation with I/O) in Table 6. These results confirm again the trends as observed in § 4.2.
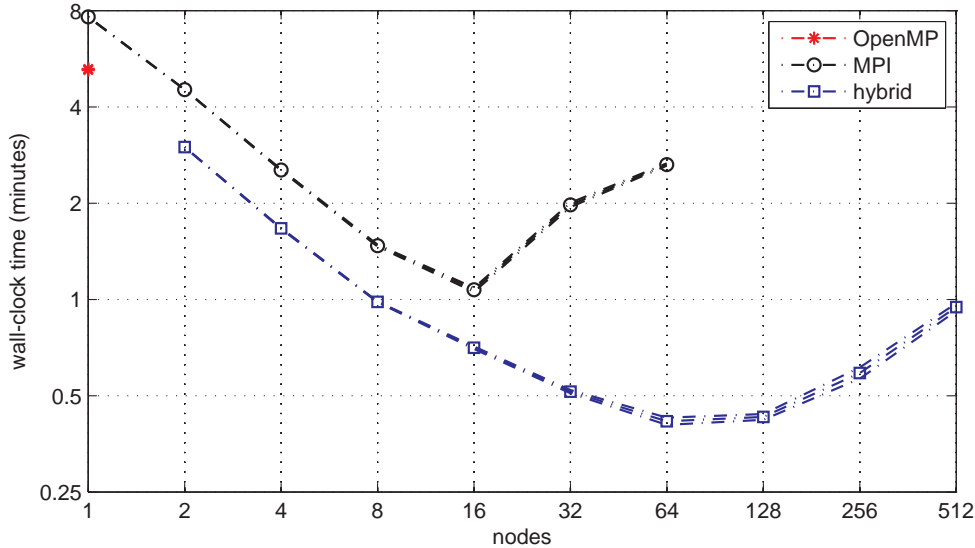
Fɪɢ. 11. *Parallel performance of the OpenMP, MPI, and hybrid versions of SWAN for the Wadden Sea case on Cartesius for large numbers of nodes. Shown is the wall-clock time in minutes for the iterations.*

4.4. Behavior for large number of nodes. We end with two numerical experiments in which we increase the number of nodes.

First, to compare MPI and hybrid implementations, we consider the Wadden Sea case (iterations, no I/O) on Cartesius. So in fact we extend Table 5 to larger number of nodes. Per node all 24 cores are used and MPI processes and/or OpenMP threads are pinned compact as described in § 4.3. Fig. 11 shows the resulting wall-clock times in minutes for the OpenMP, MPI, and hybrid versions. It can be seen that the gap between the MPI and hybrid version stays constant up to 16 nodes. From this point on the wall-clock time for the MPI version increases as the strips are becoming very thin. The MPI version divides the computational work in only one direction. For this it chooses the direction with most computational grid points, for the Wadden Sea case with 2280×979 computational grid this is the first grid direction. For 32 nodes the computational work is divided in $32 \times 24 = 768$ strips whereas there are only 2280 grid points in this direction, resulting in only 2 to 3 grid points per strip. For 128 nodes there are not enough grid points anymore to have at least one grid point per strip, therefore SWAN crashes. For the hybrid version, however, the number of MPI processes is a factor 24 lower and the OpenMP threads work in the other grid direction. In Fig. 11 the wall-clock time for the hybrid version still decreases after 16 nodes. The lowest value occurs between 64 and 128 nodes. After 128 nodes the wall-clock time increases again. In case of 95 nodes the hybrid version has strips of width 2280 / 95 = 24. Then, with 24 OpenMP threads per node, precisely all points in the computational grid for which new values can be computed simultaneously are processed on the same moment. So for this situation we obtain the maximal parallelism that we can obtain for the given algorithm (see § 3). This corresponds with the observation that the lowest value of the wall-clock time occurs between 64 and 128 nodes.

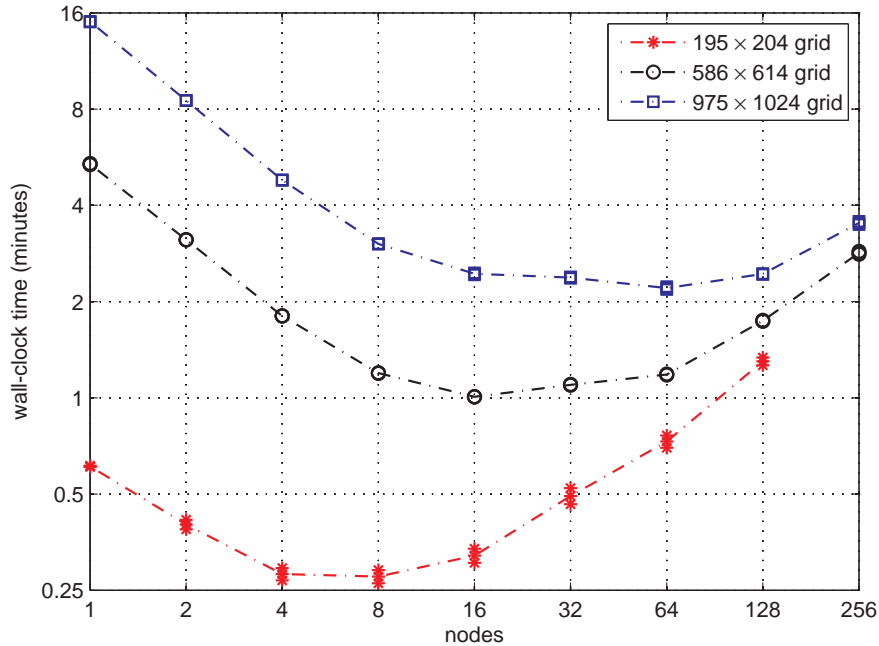Second, to study the effect of increasing the grid size, we consider the Lake Marken

FIG. 12. *Parallel performance of the hybrid version of SWAN for the Lake Marken case on Cartesius for large numbers of nodes and different grid sizes. Shown is the wall-clock time in minutes for the iterations.*

case (iterations, no I/O) on Cartesius. For this purpose the original 195×204 curvilinear computational grid of [4, 6] is uniformly refined with a factor of 3, respectively 5, in both horizontal grid directions. This resulted in a 586×614 and 975×1024 curvilinear computational grid. The corresponding number of computational grids points is 39780 (original grid), 359804 (3×3 refined grid), and 998400 (5×5 refined grid). (For the Wadden Sea case these numbers are $2280 \times 979 = 2232120$.) The study of the effect of the refinement on computational times of SWAN is important as the original grid is quite coarse for local ecological impact assessments like in [6]. We did not take into account the coupling with the shallow water solver nor the advection diffusion solver as we only want to know a lower bound of the contribution of SWAN to the computational times. Furthermore we restricted the simulation period to 1 day (instead of a typical full simulation period of 373 days). Fig. 12 shows the resulting wall-clock times in minutes for the different grid sizes. As can be seen the behavior is similar as for the Wadden Sea case in Fig. 11. For larger grids, more nodes can be used to lower computational times. Again, lowest values of the wall-clock time occur for the maximal parallelism that we can obtain for the given algorithm: for 204 / 24 ≈ 9 nodes, 614 / 24 ≈ 26 nodes, and 1024 / 24 ≈ 43 nodes (for 195×204, 586×614, and 975×1024 grid, respectively).

5. Conclusions and outlook. Because of the importance for real life applications in the Netherlands, we investigated the parallel efficiency of the current MPI and OpenMP versions of SWAN. In this paper we proposed a hybrid version of SWAN that naturally evolves from these versions. It combines the efficiency of the OpenMP version with the capability of the MPI version to use more nodes. We described the

numerical algorithm and showed its potential for important applications. Numerical experiments showed that the hybrid version improves the parallel performance of the current MPI version even more for larger number of cores per node and/or larger number of nodes. Given the current trends in hardware this is of great importance. Parallel I/O will be subject of future research.

REFERENCES

[1] N. Booij, R. C. Ris, and L. H. Holthuijsen. A third-generation wave model for coastal regions, part i, model description and validation. *J. Geoph.Research*, 104(C4):7649–7666, 1999. (Software (GNU GPL) can be downloaded from http://swanmodel.sourceforge.net).

[2] T. Campbell, V. Cazes, and E. Rogers. Implementation of an important wave model on parallel architectures. In *Oceans 2002 MTS/IEEE Conference*, pages 1509–1514. IEEE, 2002. Online at http://www7320.nrlssc.navy.mil/pubs/2002/Campbell.etal.pdf.

[3] T. C. Chan and H. A. van der Vorst. Approximate and incomplete factorizations. In *Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering*, volume 4, pages 167–202. Kluwer Academic, 1997.

[4] J. Donners, M. Genseberger, H. R. A. Jagers, C. X. O. Thiange, H. M. Schaap, P. M. A. Boderie, A. Emerson, M. Guarrasi, T. de Kler, and M. van Meersbergen. Using high performance computing to enable interactive design of measures to improve water quality and ecological state of lake Marken. In *Proceedings 15th World Lake Conference*, 2014. Online at http://www.unescowaterchair.org/activities/publications.

[5] OpenMP Forum. OpenMP application interface, version 2.5. [online], May 2005. http://www.openmp.org.

[6] M. Genseberger, R. Noordhuis, C. X. O. Thiange, and P. M. A. Boderie. Practical measures for improving the ecological state of lake Marken using in-depth system knowledge. *Lakes & Reservoirs: Research & Management*, 21(1):56–64, 2016.

[7] M. Genseberger, A. J. Smale, and H. Hartholt. Real-time forecasting of flood levels, wind driven waves, wave runup, and overtopping at dikes around Dutch lakes. In *2nd European Conference on FLOODrisk Management*, pages 1519–1525. Taylor & Francis Group, 2013.

[8] J. Groeneweg, J. Beckers, and C. Gautier. A probabilistic model for the determination of hydraulic boundary conditions in a dynamic coastal system. In *International Conference on Coastal Engineering (ICCE2010)*, 2010.

[9] W. Gropp, S. Huss-Ledermann, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI: The Complete Reference Vol. 2*. MIT Press, 1998.

[10] S. H. De Kleermaeker, M. Verlaan, J. Kroos, and F. Zijl. A new coastal flood forecasting system for the Netherlands. In *Hydro12 Conference*. Hydrographic Society Benelux, 2012. Online at http://proceedings.utwente.nl/246.

[11] R. C. Ris, L. H. Holthuijsen, J. M. Smith, N. Booij, and A. R. van Dongeren. The ONR test bed for coastal and oceanic wave models. In *International Conference on Coastal Engineering (ICCE2002)*, 2002.

[12] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference Vol. 1, The MPI Core*. MIT Press, 1998.

[13] H. L. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM J. Numer. Anal.*, 5:530–558, 1968.

[14] A. J. van der Steen. Overview of recent supercomputers. Technical report, 2008, 2010, 2011, 2012. Online at http://www.euroben.nl/reports.php. 2010 version appeared in J. J. Dongarra and A. J. van der Steen. High-performance computing systems. *Acta Numerica*, 21:379–474, 2012.

[15] M. Zijlema. Parallelization of a nearshore wind wave model for distributed memory architectures. In *Parallel Computational Fluid Dynamics - Multidisciplinary applications*, pages 207–214. Elsevier Science, 2005.