Hybrid SWAN for fast and efficient practical wave modelling - part 2[*]

Menno Genseberger[1] and John Donners[2]

[1] Deltares, P.O. Box 177, 2600 MH Delft, The Netherlands
`Menno.Genseberger@deltares.nl`
[2] Atos, Munich, Germany

Abstract. In the Netherlands, for coastal and inland water applications, wave modelling with SWAN on structured computational grids has become a main ingredient. However, computational times are relatively high. Benchmarks showed that the MPI version of SWAN is not that efficient as the OpenMP version within a single node.
Therefore, in a previous paper [5] a hybrid version of SWAN was proposed for computations on structured computational grids. It combines the efficiency of the OpenMP version on shared memory with the capability of the MPI version to distribute memory over nodes. In the current paper we extend this approach by an improved implementation, verification of the model performance with a testbed, and extensive benchmarks of its parallel performance. With these benchmarks for important real life applications we show the significance of this hybrid version. We optimize the approach and illustrate the behavior for larger number of nodes. Parallel I/O will be subject of future research.

Keywords: wave modelling · hybrid method · distributed and shared memory.

1 Introduction

In the Netherlands, for assessments of the primary water defences (for instance [8]), operational forecasting of flooding [7, 9], 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]. These are applications with SWAN on structured computational grids. 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.

Therefore, in a previous paper [5] we studied the current MPI version [13] and OpenMP version [2] of SWAN. That study was the basis of a hybrid version of SWAN.

[*] We acknowledge the Dutch Ministry of Infrastructure and the Environment, PRACE, Fortissimo, and NWO for supporting a part of this research. We acknowledge PRACE for awarding us access to resource Cartesius based in The Netherlands at SURFsara.

In the present paper (”part 2”), with results of extensive benchmarks for important real life applications, we show the significance of the hybrid version. For a proper understanding of the underlying principles we recapitulate the main ingredients from [5] of SWAN and its parallel implementations in § 2 and § 3, respectively. The setup of the benchmarks is outlined in § 4. In § 5 we indicate how we verified the model performance of SWAN for the hybrid version. Then we optimize the approach in § 6. § 7 ends with two illustrations of the behavior of the hybrid version for larger number of nodes. Parallel I/O will be subject of future research.

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 [12]. 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 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:

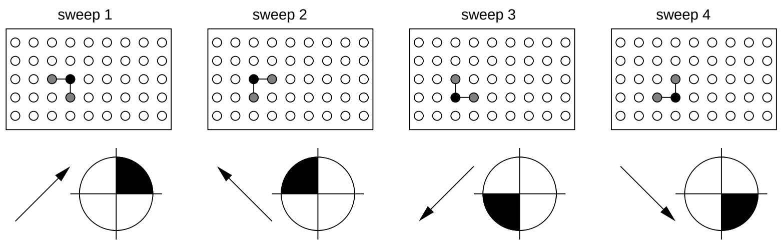
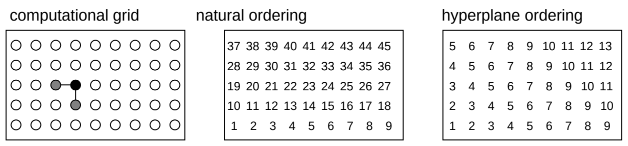


Fig. 1. Illustration of the sweep technique for the four quadrants. For every quadrant the arrow indicates the sweep direction and the black bullet represents a computational grid point that is being processed for which information comes from the two grey bullets via the upwind coupling stencil.



natural ordering

37	38	39	40	41	42	43	44	45
28	29	30	31	32	33	34	35	36
19	20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9

hyperplane ordering

5	6	7	8	9	10	11	12	13
4	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9

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.

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 [13] 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 of [13] 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

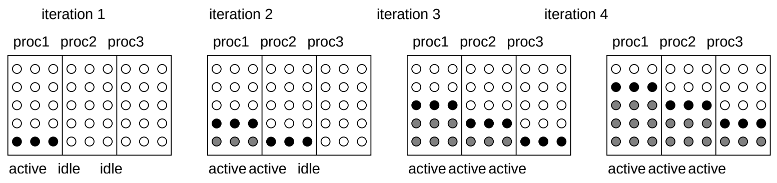


Fig. 3. Illustration of the block wavefront approach for sweep 1 of the sweep technique. Shown are succeeding iterations in case of three parallel processing units (proc1, proc2, and proc3). The black bullets represent computational grid points that are being updated in the current iteration. The grey bullets were updated in a previous iteration.

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. 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.

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, [11, § 2.4, 2012] with OpenMP.

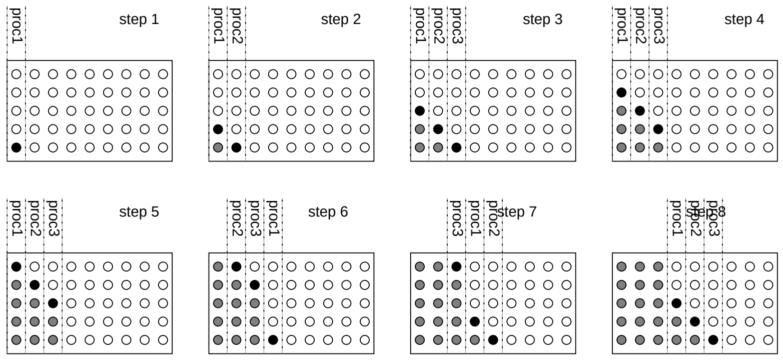


Fig. 4. Illustration of the pipelined parallel approach based on the hyperplane ordering for sweep 1 of the sweep technique. Shown are succeeding steps in case of three parallel processing units (proc1, proc2, and proc3). The black bullets represent computational grid points that are being updated in the current step. The grey bullets were updated in a previous step.

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, the parallelization strategy is of type (2): all computations can be performed without changing the original serial numerical algorithm. Therefore, except for rounding errors, the hybrid version gives identical results to 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

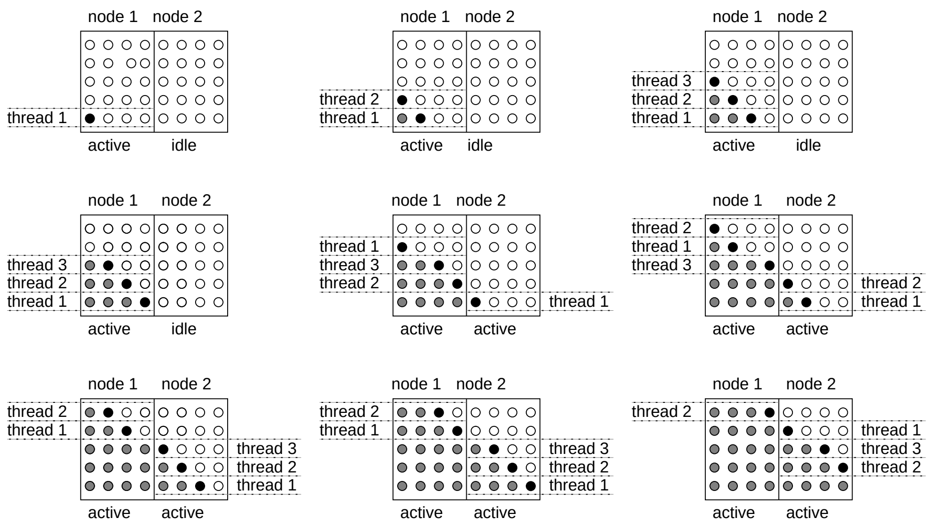


Fig. 5. Illustration of the hybrid approach based on a combination of the block wavefront approach and the pipelined parallel approach for sweep 1 of the sweep technique. Shown are succeeding steps in case of three OpenMP threads (thread 1, thread 2, and thread 3) within two MPI processes (node 1 and 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.

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. To make the hybrid version available for general use, we handed these modifications to the maintainers of the official SWAN version at Delft University of Technology [1].

4 Setup of benchmarks

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]. 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. See Fig. 6 in [5] for the bathymetry. The model is relatively large compared to other SWAN models, 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 Lake IJssel (454 × 626 rectangular computational grid with full simulation period and I/O) and Lake Marken (195 × 204, 586 × 614, and 975 × 1024 curvilinear computational grids with shortened simulation period and no I/O). See Fig. 6 in [5] for the locations. These models are incorporated too as they differ in size and concern other important application areas. The first model has been developed for operational forecasting of flooding near the Dutch major lakes [7]. The second 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].

Here we present results of benchmarks that were performed on “2690 v3” nodes of the Cartesius supercomputer (Mellanox ConnectX-3 InfiniBand adapter providing 4 × FDR resulting in 56 Gbit/s inter-node bandwidth, Intel MPI, Bull B720 bullx system, SURFsara, the Netherlands). Each node contains 2 Intel twelve-core Xeon E5-2690 v3 processors resulting in 24 cores per node with 2.60 Ghz per core. There is 30 MB cache per processor, no hyperthreading is used.

Benchmarks have been performed for MPI, OpenMP, and hybrid implementations of Deltares[3] SWAN versions 40.72ABCDE (Wadden 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 level 2 optimization for the Intel Fortran 14 compiler as used on the Intel processors.

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

[3] This Deltares version is in use for the applications mentioned in § 1. It has some small but subtle additional functionalities compared to the official version at Delft University of Technology (see website [1]) to enable interaction with a shallow water solver and wave growth in depth-limited situations like Lake IJssel and Lake Marken.

5 Verification of model performance

As mentioned before, the MPI, OpenMP, and hybrid versions do not change the original serial numerical algorithm for parallelization. Therefore, except for rounding errors, these versions give identical results to the original serial numerical algorithm, i.e. the model performance of SWAN stays the same.

During the benchmarks we verified this aspect by checking for all benchmark cases that the different combinations (MPI version, OpenMP version, hybrid version, hardware, number of processes / threads) show the same convergence behavior of the numerical algorithm of SWAN. Furthermore, for the hybrid implementation of Deltares SWAN version 40.91AB.8 the Deltares SWAN testbed was run to verify this aspect too for all testcases in the testbed. The Deltares SWAN testbed originates from the ONR testbed for SWAN [10]. 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.

6 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. Therefore, here we first try to further optimize the MPI, OpenMP, and hybrid version before we scale up to larger number of nodes. Note that, by doing so, no new discoveries are expected, the only aim is to have a good basis on the hardware before scaling up. For this purpose, with some numerical experiments we investigate the effect of the position of the computational processes on specific locations (cores, processors) inside a node. Considered are the iterations of the Wadden Sea case (i.e. no I/O).

Fig. 6 shows the parallel performance of the serial, MPI, and OpenMP versions on one Cartesius 2690 v3 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 each other 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

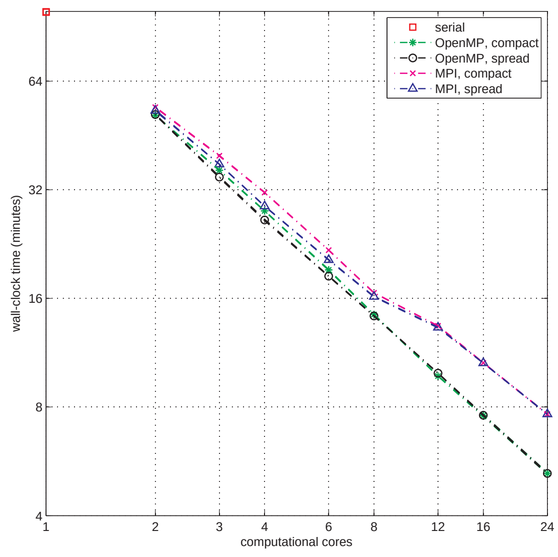


Fig. 6. Parallel performance of the serial, MPI, and OpenMP versions of SWAN for the Wadden Sea case on one Cartesius 2690 v3 node as a function of the number of cores used. For the MPI and OpenMP version two ways of pinning the processes/threads to the cores are shown: compact and spread. See § 6 for further explanation. Shown is the wall-clock time in minutes for the iterations.

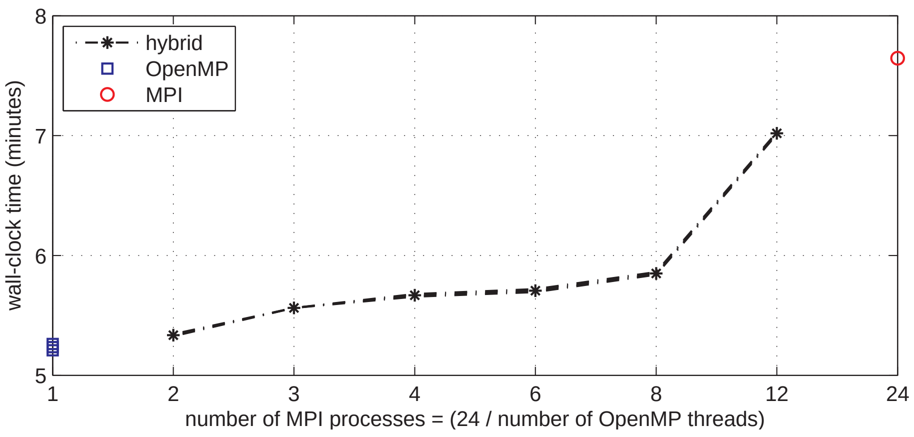


Fig. 7. Parallel performance of the hybrid version of SWAN for the Wadden Sea case on one Cartesius 2690 v3 node using all 24 cores as a function of the number of MPI processes. Each MPI process uses (24 / number of MPI processes) OpenMP threads. See § 6 for further explanation. Shown is the wall-clock time in minutes for the iterations.

20, respectively. From the figure we may conclude that, as might be expected, when using not all cores inside a node, it is beneficial for the wall-clock time to "spread" the computational processes over the cores. Note that differences between "compact" and "spread" are not that large, this may be explained from the intensive computations of SWAN that dominate in the wall-clock time (see § 2). Furthermore, in Fig. 6 it can be seen that this effect disappears when using more than 8 cores and that then also the difference between MPI and OpenMP becomes more prominent.

A similar observation can be made from Fig. 7. This figure shows the parallel performance of the hybrid version of SWAN on one Cartesius 2690 v3 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 2960 v3 node, optimal settings for the MPI, OpenMP, and hybrid versions is pinning the MPI processes and/or OpenMP threads compact to the cores. We will use these settings for the remainder of this paper.

Table 1. Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top), 2 nodes (middle), and 4 nodes (bottom) for Cartesius 2690 v3 nodes 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

With this knowledge / settings we extend the numerical experiments in § 4.2 of [5] with results on 1, 2, and 4 Cartesius nodes for the Wadden Sea case (iterations, no I/O) in Table 1 and the Lake IJssel case (full simulation with I/O) in Table 2. These results confirm the trends observed in the previous paper (the same cases are used in the benchmarks there) that the hybrid version improves the parallel performance of the current MPI version for larger number of cores per node and/or more nodes.

Table 2. Wall-clock time in minutes of the MPI, OpenMP, and hybrid versions on 1 node (top), 2 nodes (middle), and 4 nodes (bottom) for Cartesius 2690 v3 nodes 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

7 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 2690 v3 nodes. So in fact we extend Table 1 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 § 6. Fig. 8 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. In that case, the current software implementation of the MPI version (as provided by the maintainers of the official SWAN version at Delft University of Technology [1]) 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. 8 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 at the same time. So for this situation we obtain the maximal parallelism that we can obtain for the given algorithm (see also § 3 from [5]). 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 on the parallel performance of the hybrid version of SWAN, we consider the Lake Marken case (iterations, no I/O) on Cartesius 2690 v3 nodes. 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 39 780 (original grid), 359 804 (3 × 3 refined grid), and 998 400 (5 × 5 refined grid). (For the Wadden Sea case these numbers are 2280 × 979 = 2 232 120.) 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. 9 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. 8. 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).

8 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 for computations on structured computational grids. In a previous paper [5] 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.

In the current paper we extended this approach. With extensive benchmarks for important real life applications we showed the significance of this hybrid version. We optimized the approach. 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. Booij, N., Ris, R.C., Holthuijsen, L.H.: 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)

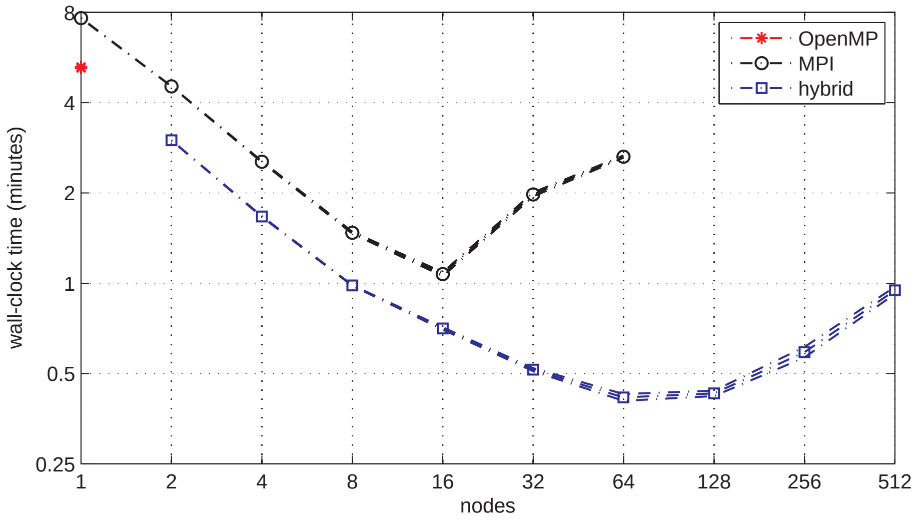


Fig. 8. Parallel performance of the OpenMP, MPI, and hybrid versions of SWAN for the Wadden Sea case on Cartesius 2690 v3 nodes for large numbers of nodes. Shown is the wall-clock time in minutes for the iterations.

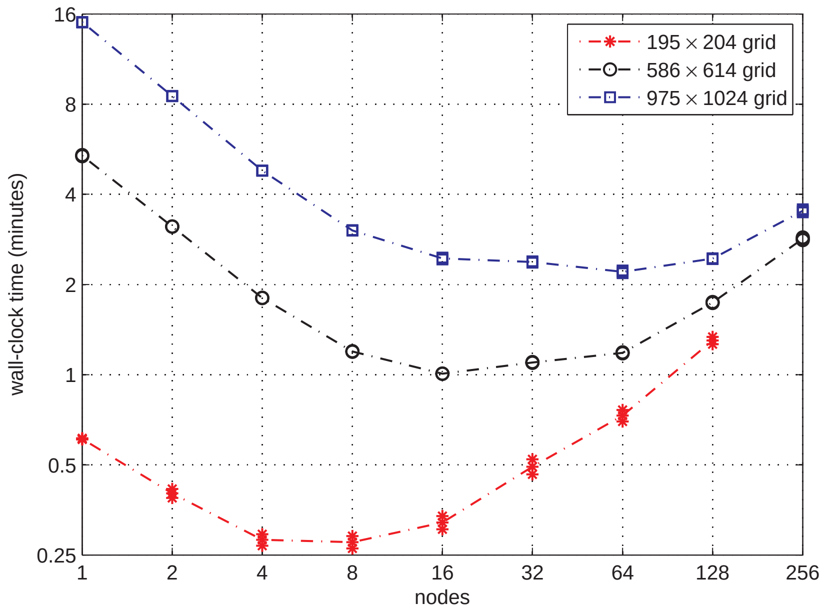


Fig. 9. Parallel performance of the hybrid version of SWAN for the Lake Marken case on Cartesius 2690 v3 nodes for large numbers of nodes and different grid sizes. Shown is the wall-clock time in minutes for the iterations.

2. Campbell, T., Cazes, V., Rogers, E.: Implementation of an important wave model on parallel architectures. In: Oceans 2002 MTS/IEEE Conference. pp. 1509–1514. IEEE (2002), online at http://www7320.nrlssc.navy.mil/pubs/2002/Campbell.etal.pdf
3. Chan, T.C., van der Vorst, H.A.: Approximate and incomplete factorizations. In: Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering. vol. 4, pp. 167–202. Kluwer Academic (1997)
4. Donners, J., Genseberger, M., Jagers, H.R.A., Thiange, C.X.O., Schaap, H.M., Boderie, P.M.A., Emerson, A., Guarrasi, M., de Kler, T., van Meersbergen, M.: 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. Genseberger, M., Donners, J.: A hybrid SWAN version for fast and efficient practical wave modelling. In: Procedia Computer Science. vol. 51, pp. 1524–1533 (2015)
6. Genseberger, M., Noordhuis, R., Thiange, C.X.O., Boderie, P.M.A.: 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. Genseberger, M., Smale, A.J., Hartholt, H.: 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. pp. 1519–1525. Taylor & Francis Group (2013)
8. Groeneweg, J., Beckers, J., Gautier, C.: A probabilistic model for the determination of hydraulic boundary conditions in a dynamic coastal system. In: International Conference on Coastal Engineering (ICCE2010) (2010)
9. Kleermaeker, S.H.D., Verlaan, M., Kroos, J., Zijl, F.: A new coastal flood forecasting system for the Netherlands. In: Hydro12 Conference. Hydrographic Society Benelux (2012), online at http://proceedings.utwente.nl/246
10. Ris, R.C., Holthuijsen, L.H., Smith, J.M., Booij, N., van Dongeren, A.R.: The ONR test bed for coastal and oceanic wave models. In: International Conference on Coastal Engineering (ICCE2002) (2002)
11. van der Steen, A.J.: Overview of recent supercomputers. Tech. rep., NWO-NCF (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
12. Stone, H.L.: Iterative solution of implicit approximations of multidimensional partial differential equations. SIAM J. Numer. Anal. **5**, 530–558 (1968)
13. Zijlema, M.: Parallelization of a nearshore wind wave model for distributed memory architectures. In: Parallel Computational Fluid Dynamics - Multidisciplinary applications. pp. 207–214. Elsevier Science (2005)