ELSEVIER

Parallel Computing 28 (2002) 35–52

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Applications

# Distributed-memory concepts in the wave model WAVEWATCH III ☆

Hendrik L. Tolman *

*SAIC/GSO at NOAA/NCEP, Environmental Modeling Center, 5200 Auth Road, Room 209, Camp Springs, MD 20746, USA*

Received 4 January 2001; received in revised form 14 June 2001; accepted 31 August 2001

**Abstract**

Parallel concepts for spectral wind-wave models are discussed, with a focus on the WAVEWATCH III model which runs in a routine operational mode at NOAA/NCEP. After a brief description of relevant aspects of wave models, basic parallelization concepts are discussed. It is argued that a method including data transposes is more suitable for this model than conventional domain decomposition techniques. Details of the implementation, including specific buffering techniques for the data to be communicated between processors, are discussed. Extensive timing results are presented for up to 450 processors on an IBM RS6000 SP. The resulting model is shown to exhibit excellent parallel behavior for a large range of numbers of processors. © 2002 Elsevier Science B.V. All rights reserved.

*Keywords:* Ocean wind-wave modelling; Distributed memory computing; Message passing

## 1. Introduction

For several decades, numerical wind-wave models have been an integral part of weather prediction at weather forecast centers around the world. Major meteorological centers now rely on the so-called third-generation wave models like WAM [3,9], or WAVEWATCH III [7,8]. In such models, all physical processes describing wave growth and decay are parameterized explicitly. Compared to previous first- and second-generation models, which parameterized integral effects of the physics rather

☆ OMB contribution No. 201.

* Tel.: +1-301-763-8133; fax: +1-301-763-8545.
*E-mail address:* hendrik.tolman@noaa.gov (H.L. Tolman).

0167-8191/02/$ - see front matter 
PII: S0167-8191(01)00130-2

than the physics itself, this is computationally expensive due to the explicit calculation of nonlinear wave–wave interactions, and due to the relatively small time-steps required by third-generation models. Although such models are still less computationally intensive than atmospheric models, they nevertheless require state-of-the-art supercomputer facilities to produce forecasts at acceptable resolutions and in a timely fashion.

The first supercomputers utilized the concept of vectorization. Such vector computers achieved increased computational performance by efficiently performing identical calculations on large sets of data. Conversion of computer models to such computers generally required systematic reorganization of the programs to generate long loop structures. Initial vector computers also required much hardware-dependent calls for basic operations. In later vector computers, additional programming was essentially limited to the inclusion of compiler directives in the source code.

The second supercomputing paradigm is that of parallelization. In this case the work is spread over multiple processors. In the simplest form (from a user's perspective), the processors share memory. As with vectorization, the success of such parallelization depends on the general structure of the program. If this structure is conducive to parallelization, modifications to the program for shared-memory parallel computers are generally small, and are usually limited to adding compiler directives to the program. Sharing memory between processors, however, requires additional logistics in the computer, which generally limits the number of processors in shared-memory parallel computers to about 16. Much more massively parallel computers with up to $O(10^3)$ processors can be constructed if the processors do not share their memory. Such distributed-memory parallel computers represent the latest development in supercomputing. Efficient application of models to such computers requires that the communication between processors becomes an integral part of the source code. Application to distributed memory computers therefore requires major code conversions, even for programs that are already applied to shared-memory parallel computers.

The present paper describes the conversion of the operational NOAA implementation of the third-generation wind-wave model WAVEWATCH III (henceforth denoted as NWW3) to a distributed memory computer architecture at the National Centers for Environmental Prediction (NCEP). The program is written in FORTRAN. For message passing between processors the message passing interface (MPI) standard has been used (e.g., Gropp [2]). In Section 2, a brief description of the model is given, together with previously used vectorization and parallelization approaches. In Sections 3 and 4, the basic distributed memory design and model modifications are discussed, as well as optimization considerations. In Section 5 the performance of the parallel code at NCEP is discussed. Sections 6 and 7 present a discussion and conclusions.

## 2. The wave model

In contrast to numerical models for the atmosphere and ocean, which provide a deterministic description of both media, wind-wave models provide a statistical

description of the sea state. The spatial and temporal scales of individual waves make it impossible to deterministically predict each individual wave for an entire ocean or sea. The random character of wind-waves makes it undesirable to deterministically model individual waves. Most statistical properties of wind-waves are captured in the distribution of wave energy over wave frequency (or wavenumber) and wave propagation direction, in the so-called wave energy density spectrum, or for short, the energy spectrum. Wave models generally predict the evolution in space and time of the energy spectrum, or alternatively, of the action spectrum. The action spectrum is the energy spectrum divided by the intrinsic frequency of the spectral components. The action spectrum is used in recent models as it allows for the transparent inclusion of effects of mean currents on the evolution of the wave field.

NWW3 predicts the evolution in the two-dimensional physical space $\mathbf{x}$ and time $t$ of the wave action density spectrum $A$ as a function of the wavenumber $k$ and direction $\theta$, as governed by the conservation equation

$$\frac{DA(k,\theta;\mathbf{x},t)}{Dt} = S(k,\theta;\mathbf{x},t). \tag{1}$$

The total derivative on the left represents the local change and effects of wave propagation. The function $S$ represents source terms for wave growth and decay, that are governed by the direct action of wind, the exchange of action between components of the spectrum due to nonlinear effects, the action loss due to white-capping, and from additional shallow water processes if applicable. Physical space ($\mathbf{x}$) and spectral space $(k,\theta)$ are discretized, and the equation is solved marching forward in time $t$, using a (global) time-step $\Delta t_g$. Below, $i, j$ and $m$ will denote discrete grid counters in $k$-, $\theta$- and $\mathbf{x}$-space, respectively. In NWW3, $\mathbf{x}$-space consists of a regular longitude–latitude grid. To reduce memory requirements of the model, spectra for grid points that are located on land are not stored, reducing $m$ to a one-dimensional counter in the two-dimensional $\mathbf{x}$-space.

To facilitate an economical solution, and at the same time to simplify the numerical approaches, Eq. (1) is solved in several consecutive fractional steps (e.g., [10]). A fractional step method is used in virtually every spectral wave model. Details, however, differ between models. In NWW3, the fractional step approach addresses spatial propagation, spectral propagation and source terms separately. Thus the following three equations are solved consecutively:

$$\frac{\partial A}{\partial t} + \nabla_{\mathbf{x}} \cdot \mathbf{c}_{\mathbf{x}} A = 0, \tag{2}$$

$$\frac{\partial A}{\partial t} + \nabla_{k,\theta} \cdot \mathbf{c}_{k,\theta} A = 0, \tag{3}$$

$$\frac{\partial A}{\partial t} = S. \tag{4}$$

Here the functional dependencies of $A$ and $S$ have been dropped for convenience. $\nabla_{\mathbf{x}}$ and $\nabla_{k,\theta}$ represent differential operators in physical and spectral spaces, respectively. $\mathbf{c}_{\mathbf{x}}$ represents the propagation velocity vector in physical space, that is a function of

the local depth $d$ and current velocity vector $\mathbf{U}$. $\mathbf{c}_{k,\theta}$ represents the propagation velocity vector in spectral space, which is a function of the spatial derivatives of $d$ and $\mathbf{U}$. The corresponding change in direction and wavenumber of individual wave groups are known as refraction and shoaling (straining), respectively. For details of these equations see [7].

In Eq. (2) the spatial propagation of a given spectral component $(k_i, \theta_j)$ is performed for all spatial grid points $\mathbf{x}_m$ simultaneously. The procedure, however, is identical for each spectral component $(k_i, \theta_j)$, and is performed independently for each $(k_i, \theta_j)$. This implies that on vector processors, the $\mathbf{x}$-space can be used for the formation of long vector loops, while simultaneously the $(k, \theta)$-space can be used for parallelisms. On multi-processor shared-memory vector machines like the Cray C90, this allowed for simultaneous vectorization and parallelization of Eq. (2) in previous versions of NWW3. Note that NWW3 defines a maximum propagation time-step $\Delta t_p$ that scales with $k$. If $\Delta t_p(k) < \Delta t_g$ (the global model time-step), propagation is performed in several steps until $\Delta t_g$ is bridged. This is relevant for the present paper as it is a potential source of load imbalances (as will be discussed later).

Conversely, in Eqs. (3) and (4) effects of refraction, straining and source terms are performed for all discrete spectral points $(k_i, \theta_j)$ simultaneously. These procedures, however, are identical for each spatial grid point $\mathbf{x}_m$, and are performed independently for each $\mathbf{x}_m$. Thus, the $\mathbf{x}$-space can be used for parallelisms, while $(k, \theta)$-space can be used for the formation of long vector loops, as is done in the previous Cray C90 version of NWW3. The solution of Eq. (3) is obtained with a fixed time-step. For the solution of Eq. (4), however, a dynamically adjusted time-step is used, where the time-step is reduced in regions with rapid change in the local wave spectrum. This generally implies that significantly more computational effort is used in and near storm centers.

One other aspect of the wave model has relevance for the design of a distributed memory version of NWW3. Ice concentrations are used as input for NWW3. If the concentration exceeds a cut-off (typically 33% or 50%), the corresponding grid point $\mathbf{x}_m$ is (temporarily) taken out of all computations.

## 3. Parallel concepts

Effective parallel distributed-memory computing requires work and data to be distributed efficiently between the available processors. Two major parallel paradigms can be distinguished.

The first is domain decomposition. Here domains considered in the model are divided in contiguous blocks of data points to be stored and processed at individual processors. Continuity of the calculation in the decomposed domains then requires communication of boundary data between processors.

The second is transposing data. Here all data needed for a given part of the calculations are gathered in a single processor before the calculation is performed. This requires the continuous reorganization ('transposing') of the way in which data are distributed over separate processors.

Domain decomposition appears to be the more common approach, as it generally requires much less communication between processors than a data transpose. Data transposes are mostly used if domain decomposition is not an option due to the physical or numerical algorithms used in a model. Examples of such models are global spectral atmospheric models as described by Sela [6].

In WAM, domain decomposition has been used even before parallel processing came into use. The domain decomposition was introduced to allow for large models to be run on a computer with limited memory. In third-generation wave models, domain decomposition can be performed in physical space only, because the nonlinear interactions in $S$ in principle require all spectral bins $(k_i, \theta_j)$ to be available simultaneously for a given spatial grid point $\mathbf{x}_m$. An example of domain decomposition as it could be applied to NWW3, for a case with four processors numbered 0–3, is given in Fig. 1(a) for an arbitrary grid. Following the terminology used in WAM, an area assigned to a given processor will be called a block. The grey area identifies land at which no spectra are stored or calculations are performed.

Domain decomposition is a standard parallel approach. The MPI standard even incorporates routines to set up the necessary communication between processors. Communication is needed only to assure that spatial propagation is properly dealt with across boundaries between blocks. Such so-called halo areas are indicated in Fig. 1(a) by the hatched boxes. The extension of a halo area depends on the propagation scheme used. The minimum extension (as shown in the figure) is one grid point at each side of the boundary. This extension allows for a first-order scheme to operate across the boundary, or for the introduction of a local first-order internal boundary condition. NWW3 uses a third-order ULTIMATE QUICKEST scheme of Leonard [4,5], which requires a halo area with an extension of two grid points on each side of the block boundaries.

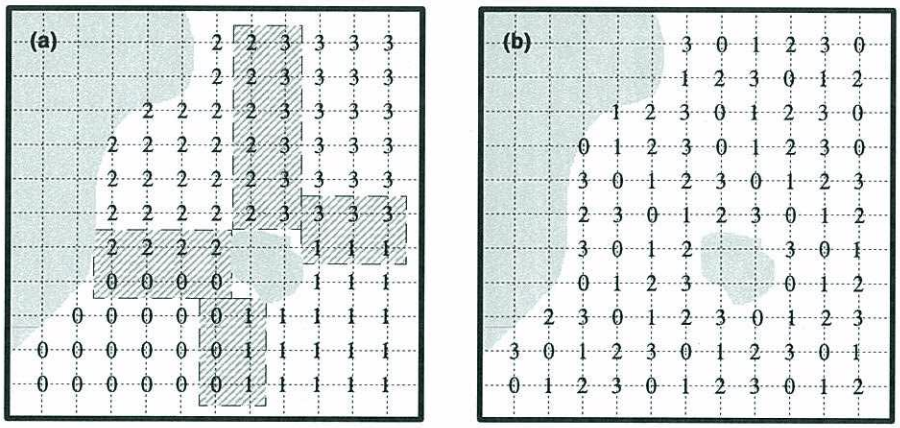


Fig. 1. Basic parallel paradigms for third-generation spectral wave models: (a) domain decomposition; (b) data scattering according to Eq. (5) and data transposes. Grey area identifies land. Four processors numbered 0–3. Number at grid line intersection identifies processor at which spectral data are stored and/or processed. Hatched area in (a) identifies minimum size of halo areas.

The advantage of domain decomposition is that it is usually relatively simple to implement, particularly when the standard MPI routines can be used. Furthermore, if the extension of the halo area is small, the amount of data to be communicated between processors is relatively small. A disadvantage is the potential load imbalances per block. Load balancing implies that the amount of work done per processor is approximately the same, particularly avoiding excessive work for a few processors. This is important in a parallel environment, as generally all processors have to wait until the last processor is finished with its work. Hence the processor with the heaviest load determines the overall model efficiency.

A simple way to implement domain decomposition is to divide the **x**-space in regular rectangular blocks. For wave models this leads to load imbalance, as the number of sea points will vary per such block as land masses (without model grid points) are always unevenly distributed over blocks. This can be alleviated by designing the blocks in such a way that all blocks include approximately the same number of sea points, as illustrated in Fig. 1(a). Such a domain decomposition becomes more complicated, but its implementation is still only a matter of proper bookkeeping.

For the particular design of NWW3, two additional load-balancing issues occur. First, the dynamical time integration used for the source terms in Eq. (4) assigns additional computational effort to areas of rapid spectral changes. Such areas typically occur in storms and are therefore spatially coherent. Such areas will usually cover only a small number of blocks, and hence result in a significant load imbalance. Because such areas with increased computational effort change dynamically, this load imbalance cannot be incorporated in the domain decomposition in a simple way. Similarly, areas covered with ice are taken out of the computations. These areas are also spatially coherent, and therefore by definition unevenly distributed over blocks. Taking ice-covered grid points out of the calculation is therefore expected to similarly impact load balancing.

Considering the above disadvantages of domain decomposition, an alternative has been considered. The load imbalance issues described above occur if Eqs. (3) and (4) are applied to coherent areas in **x**-space. The natural way to avoid this is to assign these calculations (and therefore the data storage) for each $\mathbf{x}_m$ to individual processors in a non-coherent way. Removing the spatial coherence in the data storage makes it impossible to solve Eq. (2) for parts of **x**-space at each processor, as is done in a domain decomposition. The only alternative is to gather data for all $\mathbf{x}_m$ and for a given $(k_i, \theta_j)$ in a single processor to perform calculations. In other words, this will require a full data transpose.

For comparison with domain decomposition, the data distribution of full spectra for non-coherent groups of $\mathbf{x}_m$ will be considered the 'normal' data distribution, and the corresponding spectra will be considered 'native' to the processor on which these calculations are performed. The data distribution required for solving Eq. (2) will be denoted as the transposed data distribution.

The main disadvantage of this parallel approach is that it requires that all spectral data need to be moved between processors (i.e., transposed) twice per time-step; once for going from the normal to the transposed data distribution, once to go back to the normal data distribution. Based on computational effort of the Cray version of

NWW3, and based on projected data transfer rates on parallel supercomputers, it was expected that the more massive communications inherent to the data transpose would still be economically feasible. As will be shown in Section 5, this is indeed the case.

A simple way to avoid spatially coherent data distributions, and to assure that each processor deals with the same number of grid points, is to assign each $N$th grid point to the same processor, where $N$ is the total number of processors. Grid point $m$ is thus assigned to processor $n$

$$n = \mathrm{MOD}(m-1, N). \tag{5}$$

In NWW3 the grid point counter $m$ starts in the lower left corner of the grid, and works its way up line by line ignoring grid points covered by land. Together with Eq. (5), this results in a distribution of grid points over processors for $N = 4$ as illustrated in Fig. 1(b). This distribution also assures that ice-covered grid points are equally distributed over the processors, so that the reduction of work due to taking such points out of the calculations is also well balanced.

Using the data transpose method, some care has to be taken with load balancing for spatial propagation. If the spatial propagation time-step $\Delta t_{\mathrm{p}}(k)$ is less than the overall time-step $\Delta t_{\mathrm{g}}$ for at least some $k$, spectral bins should be assigned to individual processors in such a way that the number of individual propagation calculations per processor,

$$\sum_{i,j \text{ at } n} 1 + \mathrm{INT}\left(\frac{\Delta t_{\mathrm{g}}}{\Delta t_{\mathrm{p}}(k_i)}\right) \tag{6}$$

is approximately equal for all processors $n$.

In summary, domain decomposition, which may be considered as the standard way of parallelizing codes, is applicable to ocean wave models. For NWW3, however, a data transpose method is expected to result in better load balancing, while the increased communication between processors is not expected to be important. Therefore, the data transpose method has been selected for NWW3.

## 4. Implementation and optimization

In discussing the implementation and optimization of the distributed memory version of NWW3, we will first concentrate on the calculations in Section 4.1. Input and output (I/O) will be discussed in Section 4.2. Load balancing has already been discussed in the previous section. Discussion of optimization will therefore not consider load balancing again.

### *4.1. Calculations*

In the distributed memory version of NWW3, the so-called native data distribution over processors is illustrated in Fig. 1(b) and Eq. (5). Because full spectral data

for selected grid points $\mathbf{x}_m$ are available at processor $n$, Eqs. (3) and (4) can be solved for such grid points without a need for communication. All communication needed for the computations is related to spatial propagation in Eq. (2). Before spatial propagation can be performed, the spectral densities $A(k_i, \theta_j)$ for a single spectral bin $(i, j)$ over all grid points $\mathbf{x}_m$ must be brought to a target processor $n_t$ using a 'gather' operation. This gather operation consists of send commands for the necessary data at each processor $n \neq n_t$, and the corresponding receive commands for $n = n_t$. Each gather operation thus includes communication between $n_t$ and all other processors. After Eq. (2) is solved for the spectral bin $(i, j)$, a corresponding 'scatter' operation is required, with send commands for $n = n_t$ and receive commands for each $n \neq n_t$.

Standard optimization procedures of the gather and scatter processes begin with the use of the so-called non-blocking communication commands. Using such commands, sends and receives are posted, but the program continues its operation without waiting for the corresponding sends and receives to be completed. Later in the program, the communication still needs to be finished with a 'wait' command. Furthermore, the required communications are identical for each time-step. This allows for the use of 'persistent' communication. First, send and receive operations are prepared using MPI_Send_Init and MPI_Recv_Init commands as part of the model initialization. Groups of communication commands can then be executed and controlled by using MPI_StartAll and MPI_WaitAll commands. Although the use of pertinent communications proved to have a minor effect on computational efficiency of NWW3, persistent communications greatly increase the transparency of the code by grouping large numbers of communication commands into single subroutine calls.

With these considerations, the algorithm for solving Eq. (2) for a single time-step, including the necessary data transposes, becomes (ignoring initialization as described above):

1. Prepare data gathered by sending all native data to the proper processor (MPI_StartAll).
2. Prepare the corresponding receiving of all modified native data in the scatter operation (MPI_StartAll).
3. For all spectral bins $(i, j)$ for which the propagation is performed at the present processor, do the following:
   (a) Post receives for all data to be gathered for $(i, j)$ (MPI_StartAll).
   (b) Wait for receives to finish (MPI_WaitAll).
   (c) Convert one-dimensional array $\mathbf{x}_m$ to full two-dimensional array for convenience of propagating field of spectral densities.
   (d) Perform actual propagation for $(i, j)$.
   (e) Convert full propagated field to $\mathbf{x}_m$.
   (f) Post sends for scatter operation (MPI_StartAll).
   (g) Wait for sends to finish (MPI_WaitAll).
4. Finalize communications started in step 1 (MPI_WaitAll).
5. Finalize communications started in step 2 (MPI_WaitAll).

Note that:

(i) Steps 3(c) and (e) are added to increase model transparency and are not strictly necessary because they could be incorporated in step 3(d).

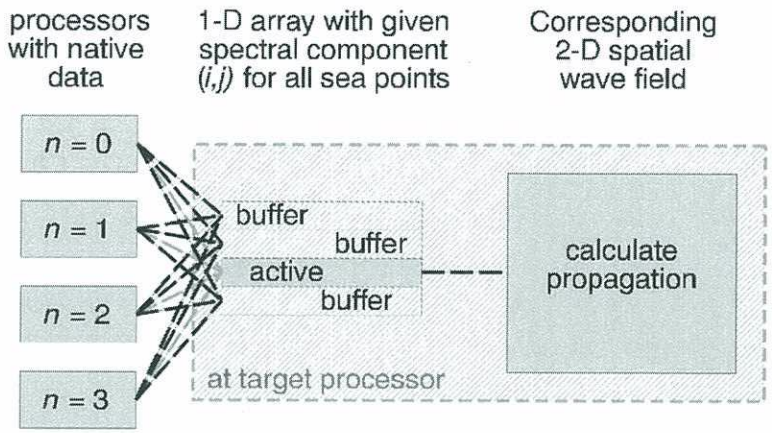


Fig. 2. Graphical representation of buffered gather and scatter operations. Thick dashed lines identify communication. Gather operations move data from left to right, scatter operations move data from right to left. The active one-dimensional array contains data for which propagation is performed. The buffer one-dimensional arrays are used for pre-gathering fields that still have to be propagated, or contain propagated fields that have not yet been scattered back to the native processors. Communication between native processors and 'active' one-dimensional array is ideally necessary only in the spin-up of the calculations.

(ii) All communication steps involving MPI consist of a single subroutine call due to the use of persistent communication.

(iii) A non-blocking scatter operation requires that this operation in fact is started in step 2 before any computation has been performed.

The above algorithm is fairly efficient, but still includes systematic wait time for communications to finish in steps 3(b) and (g), because the call to MPI_WaitAll directly follows the corresponding call to MPI_StartAll. Such wait time can be minimized or even eliminated by pre-gathering data in a buffer at $n_t$, and by similarly storing data to be scattered in a buffer before starting the actual scattering operation. Such a buffering technique is graphically depicted in Fig. 2 for $N = 4$ processors and $n_{\text{buf}} = 4$ buffer arrays. This method has been implemented in NWW3. Details can be found in its source code. [1] Note that introducing buffers increases the memory requirement of the model, and furthermore might impact data caching. This might adversely impact model performance, in particular if many buffers are used. The effect of the number of buffers on the model efficiency will be assessed in Section 5.

### 4.2. Input and output

Distributed computing also poses challenges for organizing I/O. Input to NWW3 (can) consist of wind, current, water level, temperature and ice concentration fields on the full spatial wave model grid. Because data ingest (i.e., reading of input)

[1] Available from http://polar.ncep.noaa.gov/waves/wavewatch/.

and processing are responsible for a negligible part of the computational effort of the model, and because spatial derivatives of some input fields are required within the model, each processor processes the full spatial fields of input parameters. NWW3 has five different types of output:

1. Fields of mean wave parameters on the full spatial grid (augmented with the input fields driving the wave model).
2. Full spectral data at selected output points, if necessary interpolated from the spectra at surrounding spatial grid points.
3. Full spectral data around tracks in space and time.
4. Restart files containing all spectra and some additional mean wave parameters.
5. Files with boundary data for nested models.

Mean wave parameters (output type 1) are calculated from native data on each processor, and are then gathered into a single processor that writes the data to file. Similarly, all spectra needed for the point output (output type 2) are gathered in a single processor, which performs the necessary interpolation and writes the file. A similar procedure is used for the output of boundary spectra (output type 5). To minimize model slow down due to single-processor operations in these output types, different processors are assigned the task to perform these outputs. The track output and restart files (output types 3 and 4) mostly write large numbers of spectra for either selected or all grid points. To simplify parallel implementation of this output, these files are direct access files, where each processor independently writes its own spectra to the appropriate record. This method may not be the most efficient parallel I/O solution, and requires a file system that is simultaneously accessible by all processors. It nevertheless proved adequate in the present NCEP implementations of NWW3.

## 5. Performance

The performance of the distributed memory version of NWW3 as described in the previous two sections has been tested on the phase II IBM RS6000 SP of NCEP. This machine consists of Winterhawk II nodes. Each node contains four processors, that share 2 GB of memory. The machine consists of 256 nodes (1024 processors), half of which are available for a single job due to the present queuing configuration. Although this machine represents a hybrid shared-distributed memory architecture, all processors are treated in NWW3 as if it is a pure distributed memory architecture. The MPI implementation on this machine, however, will recognize that processors are on the same node, and communicate through shared memory instead of the interconnecting network when possible. The impact of this choice will be discussed in Section 6.

As a test case, a one-day forecast is made with NCEP's operational global implementation of NWW3. The spectrum is discretized in this model using 24 directions and 25 wavenumbers (600 spectral bins). The spatial grid has a resolution of $1^\circ \times 1.25^\circ$ in latitude and longitude, and covers the globe from 78°N to 78°S, containing 30,030 sea points. The model time-step $\Delta t_g = 3600$ s. For the smallest $k$,

the propagation time-step is $\Delta t_p = 1300$ s. The smallest time-step allowed for source terms integration is set to 300 s. Additional information on this model can be found in [1], or on the NCEP/OMB waves web page. [2] The operational model uses $n_{buf} = 6$ and $N = 64$, based on previous timing results of NCEP's phase-I IBM machine. Timings are obtained directly from the executable. Such timings therefore do not include the time needed to spin up the parallel work environment on the IBM, which typically does not take a trivial amount of time, and furthermore depend on the number of processes requested.

Each individual test consists of up to three steps. First, the model is run in the so-called dry run mode, without generating model output. In dry run mode, NWW3 processes input and output, but skips all calculations. The timing for this dry run gives an estimate of the combined time required for model initialization, data ingest and time-step management. The run will be denoted as the A run, and the run time as $t_A$. In the second run (B, $t_B$), all calculations will be performed, but no output will be generated. The time difference $t_B - t_A$ is an estimate for the time spent on the actual calculations. Finally, in the C run ($t_C$), hourly grid, point and nesting output will also be generated, to assess the impact of output on parallel performance.

Two sets of tests will be performed. First, the effects of the number of buffers $n_{buf}$ as illustrated in Fig. 2 will be assessed for a selected number of processors $N$. Secondly, the parallel efficiency as a function of $N$ will be assessed using a fixed number of buffers $n_{buf}$.

To asses the impact of the buffering scheme of Fig. 2, the test case has been run with $N = 24$, 64 and 120 processors, respectively. These $N$ have been chosen somewhat arbitrarily, although $N = 64$ is particularly relevant for NCEP as this is the operational model setting. The number of buffers is increased from $n_{buf} = 1$ (no buffering), to $n_{buf} = 8$. Because the buffer is an exclusive part of the computational algorithm, only the isolated computing time $(t_B - t_A)$ will be addressed. Resulting run times and run time changes are presented in Table 1.

Each run is repeated five times to assess the reproducibility of timing results. The system on which the model is tested is known to show a small but noticeable impact on timing results of work load on the interconnecting network and on the shared file systems. Furthermore, this machine is used for both operational weather forecasting and for developmental work. The queueing system occasionally allows operational jobs to take resources from developmental work (like the present timing runs). Such occurrences are rare (about five of the 240 runs required for Table 1), and are easily identified by their anomalous timing results if multiple runs are made. Because we are interested in timing results for dedicated processors only, the above anomalous results have been removed from the present results, and have been replaced by results from additional runs. Note that the model itself is not a source of run-time variability within the sets of five calculations (the model is purely reproducible in every single aspect).

For all three numbers of processors $N$ considered, the buffering results in a speed-up of the model of about 30% for $n_{buf} \geqslant 5$. The largest incremental speedup is found when the buffer depth is increased to 2 and 3, respectively. This could be expected as

[2] http://polar.ncep.noaa.gov/waves.

Table 1
Average run times of calculational part of test case $t_r = t_B - t_A$ for several numbers of buffers $n_{buf}$ as in Fig. 2, and $N = 24$, 64 or 120 processors

| $n_{buf}$ (–) | $N = 24$ | | | $N = 64$ | | | $N = 120$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $t_r$ (s) | $\sigma_t$ (s) | $\Delta$ (%) | $t_r$ (s) | $\sigma_t$ (s) | $\Delta$ (%) | $t_r$ (s) | $\sigma_t$ (s) | $\Delta$ (%) |
| 1 | 156.1 | 1.2 | – | 67.1 | 2.1 | – | 43.2 | 2.1 | – |
| 2 | 138.2 | 0.4 | 11.4 | 56.0 | 1.2 | 16.6 | 37.9 | 4.1 | 12.2 |
| 3 | 120.9 | 2.9 | 22.5 | 49.0 | 0.5 | 26.9 | 33.4 | 2.4 | 22.7 |
| 4 | 113.0 | 1.4 | 27.6 | 49.3 | 3.0 | 26.6 | 31.9 | 3.9 | 26.2 |
| 5 | 109.6 | 1.3 | 29.8 | 47.9 | 1.4 | 28.6 | 29.8 | 1.7 | 31.1 |
| 6 | 106.7 | 0.2 | 31.7 | 45.7 | 2.1 | 31.9 | 30.8 | 1.9 | 28.7 |
| 7 | 108.1 | 2.3 | 30.8 | 47.5 | 3.2 | 29.3 | 31.0 | 2.2 | 28.3 |
| 8 | 109.8 | 1.9 | 29.6 | 47.9 | 2.2 | 28.6 | 29.4 | 1.7 | 32.1 |

5 runs per case. $\sigma_t$ represents the standard deviation of run time per case. The change $\Delta$ is the improvement relative to the case without buffering.

a minimum of three buffers are needed to avoid blocking behavior in the receive part of the gather and the send part of the scatter operation. One buffer is then actively involved in the gather operation, one in the calculation and one in the scatter operation. Although $n_{buf} = 3$ is the minimum buffer depth required to avoid blocking, it does not necessarily imply that no blocking will occur. Apparently, some additional benefit is gained from further increasing the buffer depth to $n_{buf} = 5$. Further increasing the buffer depth introduces variations in the timing results similar to the variability for the results for a given buffer depth ($\sigma_t$ in the table). Whereas there is no added benefit for increasing the buffer depth, there appears to be no penalty either. Based on similar timing results of NWW3 on the Phase-I IBM at NCEP, the default buffer depth was set to $n_{buf} = 6$. This setting will be used in the remainder of the present tests.

With the above buffer depth, the impact of the number of processors $N$ on the run time and the corresponding scalability of the model will be assessed. To this end the model has been run on a variable number of processors ranging from 1 to 450. For $N \leqslant 50$, all possible $N$ have been considered. For larger $N$, runs have been performed with a regular increase in the increment of $N$. To simplify dealing with anomalous timing results as discussed above, optimum results of 5 runs per case are considered.

The parallel performance of the model has been assessed using three parameters. The first is the overall run time for $N$ processors ($t_N$), as is shown in Fig. 3. The second parameter is a measure for the overall parallel efficiency of the model. This efficiency $\zeta$ is defined as the ratio of the measured model speed up ($t_1/t_N$) to the potential speed up $N$

$$\zeta = \frac{t_1}{t_N N}. \tag{7}$$

$\zeta = 1$ corresponds to a perfectly parallel behavior, lower values identify less-than-ideal parallel behavior. This parallel efficiency is presented in Fig. 4. The final

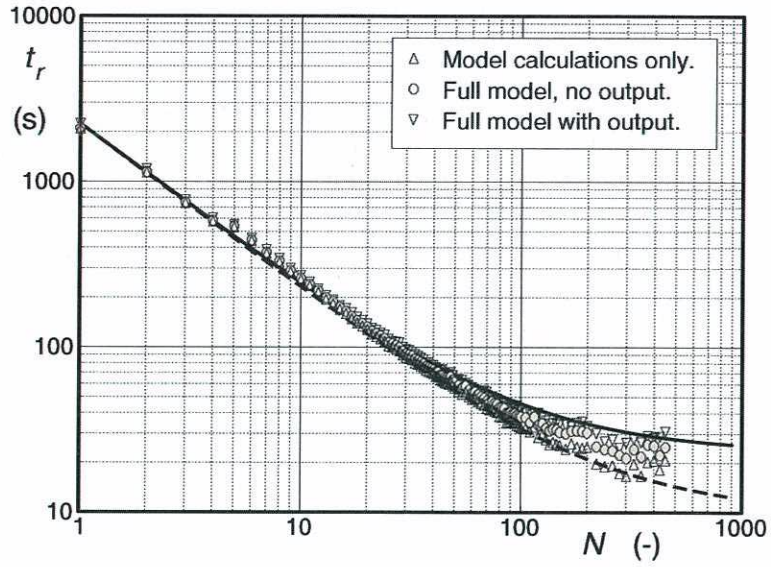


Fig. 3. Run times $t$ for the test case as a function of the number of processors $N$. Solid line: Estimate from Amdahl's law (8) with $p = 0.990$. Dashed line: Estimate from (8) with $p = 0.996$.

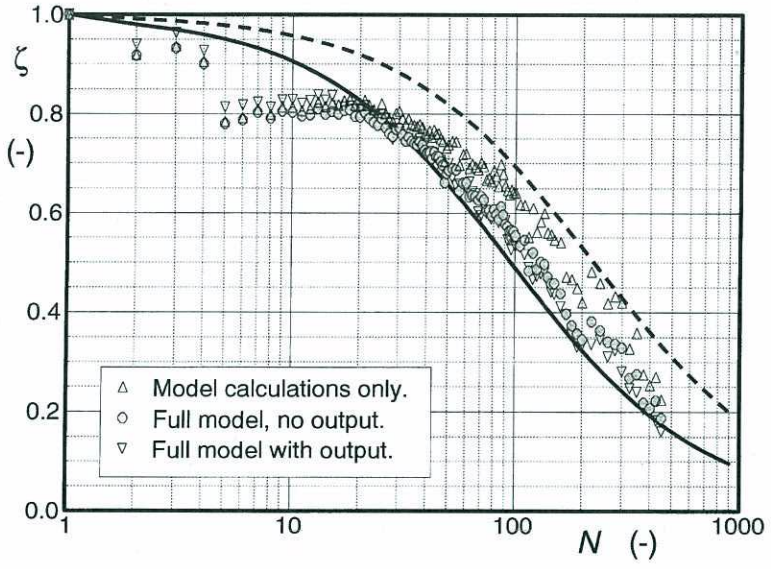


Fig. 4. Parallel efficiency $\zeta$ according to Eq. (7) estimated from run times in Fig. 3. Lines as in Fig. 3.

parameter considered is the parallel fraction $p$ of the model, as estimated using Amdahl's law. This law states that the minimum run time $t_{\min}$ for a given parallel fraction $p$ of a model is

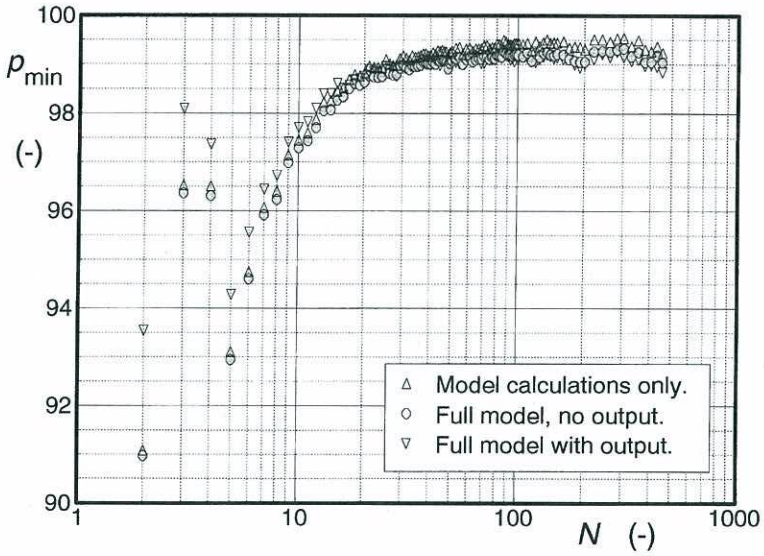


Fig. 5. Estimate of minimum parallel fraction of model $p$ from Eq. (9) estimated from run times in Fig. 3.

$$t_{\min} = (1-p)t_1 + p\frac{t_1}{N}. \tag{8}$$

Considering that $t_{\min} < t_N$, because $t_N$ consist of $t_{\min}$ plus an overhead for the explicit communication needed in a distributed memory environment, Eq. (8) can be re-written as

$$p \geqslant p_{\min} = \frac{N}{N-1}\left(1 - \frac{t_N}{t_1}\right), \tag{9}$$

where $p_{\min}$ is the minimum parallel fraction required to attain the observed run time $t_N$. This estimate for the minimum parallel fraction is presented in Fig. 5.

## 6. Discussion

Fig. 3 shows a systematic decrease of the run time $t_N$ as a function of the number of processors $N$, up to $N \approx 240$. For larger numbers of processors there appears to be little or no gain in run time, or perhaps even an increase. The latter behavior is difficult to assess due to the variability in the timing results. The three run times considered (symbols, see legend) are similar, particularly for smaller $N$. For larger $N$ their differences become noticeable, but never dominating. For larger $N$ timing results are roughly bounded by timing estimates according to Amdahl's law (8) with a parallel fraction of $p = 0.990$ (solid line) and $p = 0.996$ (dashed line).

Fig. 4 shows somewhat more complicated behavior for the parallel efficiency $\zeta(N)$. For $N \leqslant 4$ the efficiency $\zeta > 0.9$. For $N = 5$, the efficiency drops to $\zeta \approx 0.8$, where it

remains for $N$ up to approximately 30. For further increasing $N$, $\zeta$ systematically decreases, roughly equivalent to an expected decrease of $\zeta$ as estimated from Amdahl's law (8) (solid and dashed lines). As in Fig. 3, differences between the three run-time estimates are most pronounced for larger $N$, but do not differ dramatically.

Fig. 5 shows the corresponding behavior of the minimum parallel fraction $p_{\min}$ as estimated from Eq. (9). Different behaviors again can be seen for $N \leqslant 4$ and for larger $N$. For larger $N$, $p_{\min}$ systematically increases to $p_{\min} \approx 0.990$ for $N \approx 30$, after which $p_{\min}$ stays constant, or slowly increases, depending on which timing is used (compare symbols). For $N > 200$, $p$ appears to display a small but systematic decrease with $N$. Contrary to previous figures, $p_{\min}$ shows distinct differences between estimates based on the three different time definitions for small $N$, with the largest parallel fractions for the model including output ($\bigtriangledown$).

Figs. 3–5 show several distinct ranges of parallel behavior.

The first range is from $N = 1$ to 4, and is clearly bounded by the almost discontinuous behavior when increasing $N$ from 4 to 5. The high efficiencies $\zeta$ (Fig. 4) in this range are a direct consequence of the hardware used here. For $N \leqslant 4$, all processors used are located on the same node. In such conditions, the parallel environment on the IBM routes MPI communications through shared memory. Once communications are required between nodes ($N \geqslant 5$ for the present hardware), MPI communications have to go through the slower interconnections between nodes. The notable decrease of $\zeta$ for $N \rightarrow 5$ indicates that the limitation of parallel efficiency for such $N$ is for a significant part due to the overhead introduced by the explicit communications.

The second range goes from $N = 5$ to $N \approx 30$, and is characterized by a nearly constant bulk efficiency $\zeta$, and a systematically increasing minimum parallel fraction $p_{\min}$. An explanation for this behavior may be found in the mechanics of the communication, which correspond to that of an MPI_AllToAll operation. For each gather operation, [3] the target processor $n_t$ receives a message from each processor $n \neq n_t$ with a message size $M/N$, where $M$ is the total number of sea points in the model. When $N$ increases, the message size gets smaller, and the message will therefore arrive faster at its destination. As messages from different sources should show minimal interference, this implies that all data are available at the target processor earlier. Because the travel time over the interconnect network dominates communication times, this implies that the full gather operation can become more efficient if $N$ increases. The increase in $p_{\min}$ is expected to saturate if the message size gets so small that the latency in the communications dominates the actual data transfer. Note that the use of buffers reduces the impact of the efficiency of the gather operation, but does not remove it completely, as the buffering by definition will not work for the first and last field to be processed.

The third range covers $N \approx 25$–150, and is characterized by a nearly constant minimum parallel fraction $p_{\min}$ and a systematically decreasing bulk efficiency $\zeta$. The reduced bulk efficiency is a direct result of Amdahl's law (compare symbols and solid and dashed lines in Fig. 4).

[3] For each scatter operation the same holds true for the native data to be received.

A fourth range covers $N > 150$, and in many ways shows similar behavior as the third range. Specific differences are that embedded within the systematic behavior, sub-ranges can be identified. Particularly clear is the sudden increase of $\zeta$ and $p_{\min}$ for $N \approx 220$, which proved reproducible with additional calculations. A second difference is that $p_{\min}$ shows a small but systematic decrease for $N > 220$. Both behaviors can be attributed to load balancing for the solution of Eq. (2), which is expected to break down when $N$ approaches the number of discrete spectral components (600 in the present test cases). This is expected to lead to a increased sensitivity to the actual $N$, and to a reduced parallel behavior. Note that this behavior is noticeable, but does not yet have a dramatic impact on the scalability of the model for $N$ up to 450.

As mentioned above, the differences in timing results for the computational part of the model, the model without output, and the full model ($\triangle$, $\circ$ and $\triangledown$ in Fig. 3) are moderate. The mostly linear initialization of the model (difference between $\triangle$ and $\circ$ in Fig. 3) takes about 5% of the run time $t_B$ for $N = 1$, increasing to 10% for $N = 30$, 20% for $N = 100$ and 35% for $N = 450$ (figure not shown here).

Adding output (compare $\circ$ and $\triangledown$ in Fig. 3) has a similar but smaller impact, with an increase of 10% for $N = 100$ and up to 25% for $N = 450$. Attempts to further parallelize the output also have not been deemed necessary at NCEP at the present. Two additional remarks have to be made considering output. First, the model with output actually has a higher parallel fraction than the other timing results for small $N$, but a lower parallel fraction for large $N$ (symbols in Fig. 5). This can be explained because individual types of output are performed by different processors. This apparently is efficient if each processor has his own output type ($N \approx 5$), However, because there are only five output types, little is to be gained when $N > 5$, and in effect the output will become less parallel. Secondly, writing a single restart file (not included in the test case) adds approximately 30 s to the run time on NCEP's phase II IBM system. This rather slow I/O can be attributed to inefficient token passing when writing from many processors to a single direct access file. Alternatives to this method of generating restart files might be considered in the near future, but are not deemed necessary in the present NCEP environment.

The parallel approach used in NWW3 with its accompanying data transposes proved excellently suitable for massively parallel systems. For $N > 30$, the model is well over 99% parallel, as required to scale properly for large numbers of processors. For smaller numbers of processors ($N < 30$) the model is less parallel ($p_{\min} < 0.99$, Fig. 5). The bulk efficiency $\zeta$, however, remains as large as $\zeta = 0.8$ (Fig. 4), indicating that a more parallel approach can speed up the model by less than 20% for smaller numbers of processors. Comparing actual timing results with Amdahl's law in Fig. 4, in practice such a speed up could be expected to be no more than 10–15% for $N < 10$, and 5–10% for $10 < N < 20$. Hence the present approach is proven to be viable and efficient for essentially arbitrary numbers of processors.

Apart from the general parallel design, the success of the implementation in NWW3 depends on two details of implementation. First, the buffer techniques used in the data transpose appear to result in a systematic reduction of the model run time of about 30% (Table 1). Secondly, the data transposes have been implemented without any 'hard' synchronization of processors using MPI_Barrier calls. Instead, all

synchronization takes place using the logical waiting for local communication to be finished. This allows for processors to be temporarily 'out of synch', without resulting in an increased overall run time. Instrumentation of the code for timing purposes suggests that this is also an important reason for the highly parallel behavior attained.

The potential of domain decomposition for NWW3 has so far only been discussed qualitatively. For large numbers of processors $N$, individual blocks will become small. Casual inspection of the local and instantaneous time steps used in Eq. (4) for the present test case, suggest that local and instantaneous increases of computational time per block of 20–40% will be common. Such load imbalances, combined with the highly parallel behavior of the present approach, make further investigation of domain decomposition for NWW3 moot.

As discussed in the beginning Section 5, the machine on which NWW3 is run is a hybrid shared/distributed memory machine, where four processors on a node share their memory. The parallel approach of NWW3 as presented here, however, uses a strict distributed memory paradigm. Tentatively, addition gains in parallel performance could be achieved if a hybrid parallel approach is adopted for NWW3.

A previous version of NWW3 was run on a Cray C90 using a strict shared-memory parallel approach. This model reached parallel fractions $p$ of just over 98% on up to 12 processors. This code was ported to the present IBM computer and other platforms, resulting in significantly lower parallel fractions (typically 95%). Because all these configurations represent poorer parallel behavior than achieved with the present approach for massively parallel configurations (see Fig. 5, $p > 98\%$ for $N > 20$), it is not likely that a more complex hybrid parallel approach would prove more efficient than the present distributed memory approach of NWW3 on our present IBM supercomputer.

## 7. Conclusions

Parallel concepts for spectral wind-wave models have been discussed in the context of the WAVEWATCH III model as implemented at NOAA/NCEP. Conventionally, domain decomposition methods have been used for the parallelization of such models. In particular for this model, however, a more unconventional data transpose method appeared more suitable on theoretical grounds. Having implemented such a data transpose method, extensive timing and scaling tests have been performed. It was shown that the model shows excellent parallel behavior for a large range of numbers of processors on NCEP's IBM RS6000 SP supercomputer. Furthermore, it is shown that a data buffering technique used in the data transpose is essential in attaining such parallel behavior.

## Acknowledgements

The author would like to thank Jim Tuccillo and George Vandenberghe for their support during the design and testing of NWW3 on the IBM phase I and II systems

at NCEP, and D.B. Rao, Joe Sela and Mark Iredell for their comments on early drafts of this paper. The present study was made possibly by funding from the NOAA High Performance Computing and Communication (HPCC) office.

## References

[1] H.S. Chen, L.D. Burroughs, H.L. Tolman, Ocean surface waves, NWS/NCEP Tech. Procedures Bull. 453 (1999).
[2] W. Gropp, E. Lusk, A. Skjellum, Using MPI, MIT Press, Cambridge, MA, 1997.
[3] G.J. Komen, L. Cavaleri, M. Donelan, K. Hasselmann, S. Hasselmann, P.E.A.M. Janssen, Dynamics and Modelling of Ocean Waves, Cambridge University Press, Cambridge, MA, 1994.
[4] B.P. Leonard, A stable and accurate convective modelling procedure based on quadratic upstream interpolation, Comput. Methods Appl. Mech. Eng. 19 (1979) 59–98.
[5] B.P. Leonard, The ULTIMATE conservative difference scheme applied to unsteady one-dimensional advection, Comput. Methods Appl. Mech. Eng. 88 (1991) 17–74.
[6] J.G. Sela, Weather forecasting on parallel architectures, Parallel Comput. 21 (1995) 1639–1654.
[7] H.L. Tolman, User Manual and System Documentation of WAVEWATCH III Version 1.18. NOAA/NWS/NCEP/OMB Techn. Note 166, 1999. Available from http://polar.ncep.noaa.gov/waves/wavewatch.
[8] H.L. Tolman, D.V. Chalikov, Source terms in a third-generation wind wave model, J. Phys. Oceanogr. 26 (1996) 2497–2518.
[9] WAMDIG, The WAM model – a third generation ocean wave prediction model, J. Phys. Oceanogr. 18 (1988) 1775–1810.
[10] N.N. Yanenko, The Method of Fractional Steps, Springer, Berlin, 1971.