Implementation of an Important Wave Model on Parallel Architectures

Tim Campbell
Mississippi State University

John Cazes
Northrop Grumman IT

Erick Rogers
Naval Research Laboratory SSC

Mississippi State University
1103 Balch Blvd., Rm. 248
Stennis Space Center, MS, 39529, USA
tjcamp@navo.hpc.mil

Abstract- **SWAN (Simulating WAves Nearshore), developed at the Delft University of Technology, is an important third generation wave model used to simulate short-crested wind-generated waves in shallow water areas such as coastal regions and inland waters. The model solves a four-dimensional (2 spatial dimensions, wave direction, and wave frequency) spectral action balance equation using a semi-implicit upwind scheme. Relative to other less advanced wave models, SWAN is more computationally demanding, and a parallel version is necessary in order to decrease turn-around time, improve the model resolution for large coastal regions, and migrate SWAN into Navy operational use.**

In this paper we present a new parallel implementation of SWAN using a pipelined parallel approach which does not alter the order of operations in the sequential numerical algorithm. The implementation uses OpenMP compiler directives and runs on shared-memory multiprocessor computers. This approach represents a non-traditional, *i.e.*, not loop-level, way of using OpenMP. Performance measurements show that turn-around time for high-resolution model applications can be significantly reduced with the parallel implementation. The parallel implementation has been verified and model output matches "bit-for-bit" with the original sequential code for both stationary and non-stationary cases. The new parallel code has already been incorporated into the next official release of SWAN and is beginning transition into operational use.

I. INTRODUCTION

For over 30 years the Navy has been performing routine operational wave forecasts to support fleet operational and training exercises. During this time wave modeling technologies have undergone significant advances in representation of the complex dynamics that contribute to wave generation and interaction [1]. Modern wave models used in weather prediction centers around the world are known as "third-generation" models. Wave prediction for oceanic basins and deep water is performed using third-generation models, such as, WAM [2] and WAVEWATCH III [3]. The changing focus of naval warfare from global threat to regional conflict has introduced the requirement for accurate and timely wave forecasts for the littoral zone. Although the deep water models have been adapted to model the smaller, high-resolution regions, they lacked the sophistication necessary for estimating coastal wave conditions. Presently second generation model STWAVE [4], with forcing at the offshore boundary provided by WAM, is used for Navy operational wave forecasting in the high resolution coastal domains. In the near future STWAVE will be replaced by the more advanced third generation model, known as SWAN (Simulating WAves Nearshore) [5]. SWAN was developed at the Delft University of Technology under the ONR Advanced Wave Prediction Program.

The sophisticated technologies developed in SWAN have proven successful in providing accurate wave estimates in the littoral region. However, in addition to accuracy, operational wave estimates must be provided in a timely manner, otherwise, the information is useless. SWAN is more computationally demanding than STWAVE, thus, to make the model operationally viable, it is necessary to take advantage of parallel processing to reduce turn-around time. In this paper we present an implementation of SWAN for parallel computers that satisfactorily addresses the issue of turn-around time. Section II describes the SWAN model and its implementation. Section III describes the details of the parallel implementation of SWAN. Performance measurements of the parallel code, along with description of a technique for estimating performance, are detailed in Section IV. A summary is given in Section V.

II. MODEL DESCRIPTION

Thoughtful observation of the ocean, especially on a windy day, will lead one to recognize that waves on the ocean surface can be characterized as irregular and random. In SWAN this stochastic process is modeled using what is known as a phase-averaging approach which, instead of solving for the actual free surface, describes the wave conditions at a given location as a linear superposition of wave energies defined in terms of their frequency and direction. In this context, the dependent variable is the spectral density of wave action (analogous to particle dynamics). The evolution of the spectral action density is described by the action balance equation which, for Cartesian coordinates, is

$$\frac{\partial}{\partial t}N + \frac{\partial}{\partial x}c_x N + \frac{\partial}{\partial y}c_y N + \frac{\partial}{\partial \sigma}c_\sigma N + \frac{\partial}{\partial \theta}c_\theta N = \sum_{i=1}^{n} S_i . \quad (2.1)$$

The action density N is equal to the energy density divided by the wave frequency. The independent variables are the time t, the location in geographic space (x,y), and the location in spectral space (σ,θ), that is, the wave frequency σ, and the wave direction θ. The propagation velocity in each independent dimension is specified by the appropriate c_a. The source term expressed on the right-hand side of the action balance equation represents the effects of generation, dissipation, and nonlinear wave-wave interactions.

Integration of the action balance equation is implemented with finite difference schemes in all five dimensions using a constant resolution in the discretization. The numerical propagation schemes applied are implicit second-order upwind in the geographic space and implicit first-order upwind, supplemented with a central approximation, in the spectral space. The implicit nature of the geographic propagation schemes used by SWAN is the primary difference between SWAN and other third generation wave

models, e.g. WAM and WAVEWATCH III. For a chosen direction of propagation, the upwind scheme implies that the state at each geographic grid point is dependent only on the state in the upwind grid points. This permits decomposition of the spectral space into four quadrants, as shown in Fig. 1, according to the wave propagation direction. With the exception of interactions due to refraction, frequency shifting, and nonlinear source terms, the solution process in each quadrant can be carried out independently from the other quadrants. To properly account for the interactions between quadrants, the whole solution process involves iteratively computing a sequence of four forward marching sweeps across the geographic grid, as illustrated in Fig. 1, with each sweep utilizing only boundary conditions or previously calculated points in the upwind direction. The solution of a submatrix (spectral space) is required at each geographic grid point. When no currents are present and the depth is stationary the submatrix is tridiagonal and solved directly. Otherwise, the submatrix is banded and solved using an iterative method.

The SWAN implementation is full-featured with numerous user configurable options. The model can be run as either stationary or non-stationary and using either Cartesian or spherical coordinates. SWAN can be readily nested in the oceanic scale models WAM and WAVEWATCH III. The SWAN code is sequential, consisting of approximately 30,000 logical lines of code with over 200 subroutines written mostly in Fortran 77. A small number of Fortran 90 constructs have been added in recent years.

III. APPROACH

The first step in creating a parallel program from a sequential one involves decomposing the computation into tasks that can be exploited by the parallel program. In SWAN we identify as a task the computation of the state, $N(\sigma,\theta)$, at a geographic grid point. The next step is to examine the data and numerical algorithms to determine the dependencies and potential for concurrent computation. Fig. 2 illustrates the task dependencies (arrows) and available concurrency (dashed lines) for sweeps 1 and 2. The dependencies for a task arise from the upwind stencil are understood by following the arrows from tail to head. In other words, for sweep 1, the task at grid point (i,j) is dependent on the completion of the tasks at the upwind grid points $(i-1,j)$ and $(i,j-1)$. Tasks that do not share dependencies (those connected with dashed lines in Fig. 2) can be computed concurrently. These tasks lie along a diagonal that is perpendicular to the sweep direction as defined in Fig. 1.

Based on the defined tasks, their dependencies, and the identified concurrency we chose to implement a pipelined parallel approach [6]. Pipelining is commonly used in situations in which several operations must be completed in a sequence $\{O_1,\dots,O_n\}$ and those operations have the property that some steps of O_{i+1} can be carried out before operation O_i is finished. In SWAN we consider the rows of the geographic grid as the sequence of operations in which the steps of an operation consist of computing the state at each grid point in a row. The steps of a pipelined parallel approach are illustrated in Fig. 3 for sweep 1 on a 4x6 geographic grid with 3 processors. Each row of the geographic grid is assigned to a processor (P0, P1, …) in a round-robin fashion. The computation begins with the first upwind grid point; computation of the state at downwind grid points begins as dependencies are satisfied. One can think of the processors "flowing" across the grid in the downwind direction. Note that upon completing a row a processor begins computing on the next available row (see steps 5, 6, and 7 in Fig. 3). The pipelined parallel steps for sweeps 2, 3, and 4 follow a similar pattern, the difference being the starting grid point and the "flow" direction. For a sequential program a sweep on a 4x6 geographic grid requires 24 steps. From Fig. 3 we see that with the pipelined parallel approach on 3 processors the number of steps has been reduced to 10.

We implemented the pipelined parallel approach in SWAN using OpenMP compiler directives [7]. The directives produce multithreaded code that runs on shared-memory multiprocessor computers. The code modifications

Sweep 1 (0 - 90°) Sweep 2 (90 - 180°)
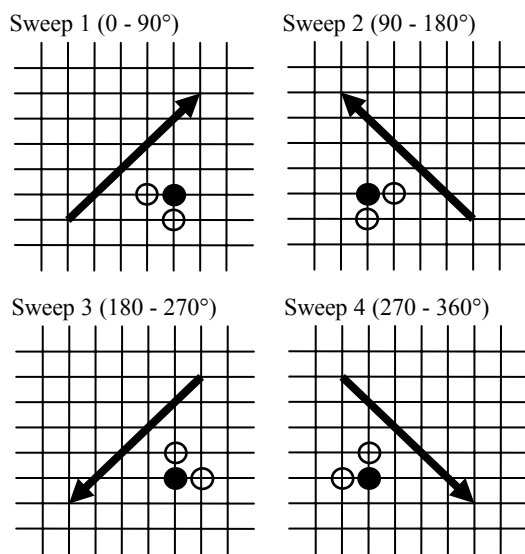
Sweep 3 (180 - 270°) Sweep 4 (270 - 360°)

Fig. 1. Sweep technique used in SWAN. The spectral space is decomposed into four quadrants based on propagation direction. The solution process for each quadrant consists of a sweep over the geographic grid in the chosen propagation direction, as indicated by the arrow. The open circles indicate the upwind grid points for a corresponding computational grid point (filled circle). Due to interactions between quadrants the complete solution is obtained iteratively, where each iteration step includes the four sweeps.
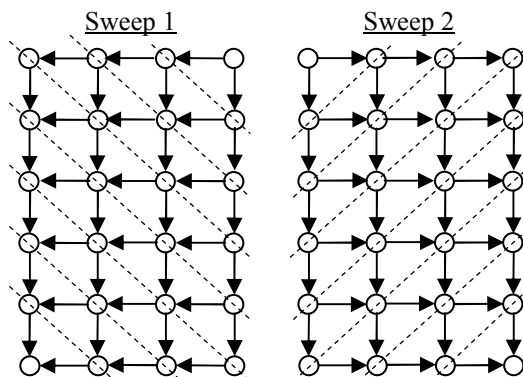

Sweep 1 Sweep 2

Fig. 2. Dependency and concurrency on a 4x6 geographic grid for sweeps 1 and 2 of SWAN. The horizontal and vertical arrows indicate dependencies; computation at a grid point is dependent on completion of computation at upwind grid points. Diagonal dashed lines connect points with no dependencies among them that can be computed in parallel.
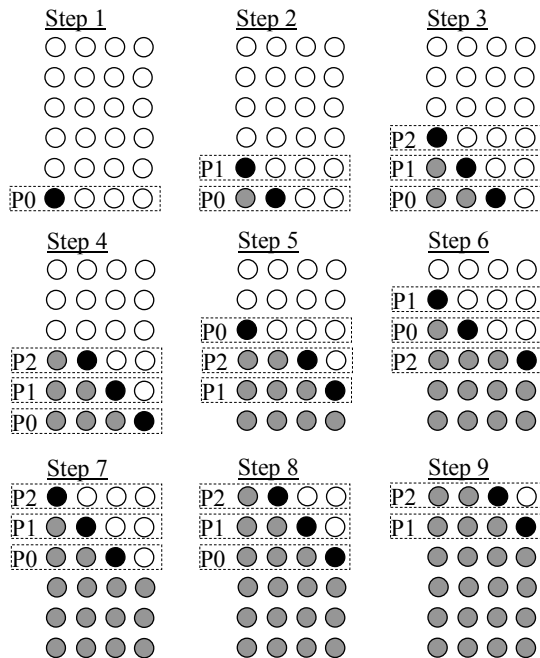
Fig. 3. Pipelined parallel steps for sweep 1 on a 4x6 geographic grid with 3 processors. Rows are assigned to processors (P0, P1, …) in a round-robin fashion, as indicated by the dashed boxes. Black circles represent grid points for which the state is computed during the current step; grey circles represent grid points for which the state is already computed. Upon completing a row a processor begins the next available row, as seen in steps 5, 6, and 7.

```
      SUBROUTINE SWCOMP
      …pre-iteration setup…
      …allocate shared work arrays…
!$OMP PARALLEL DEFAULT(SHARED)
!$OMP+PRIVATE(ITER,SWPDIR,…)
      …allocate private work arrays…
      DO ITER = 1, ITERMX
        DO SWPDIR = 1, 4
          …set sweep parameters…
          …set LLOCK array…
!$OMP DO SCHEDULE(STATIC,1)
          DO IY = IY1, IY2, -IDY
            DO IX = IX1, IX2, -IDX
              DO WHILE(LLOCK(IX,IY+IDY)
!$OMP FLUSH

              END DO
              CALL SWOMPU(…,IX,IY,…)
              LLOCK(IX,IY) = .FALSE.
            END DO
          END DO
        END DO
!$OMP BARRIER
        …check convergence…
      END DO
      …deallocate private work arrays…
!$OMP END PARALLEL
      …deallocate shared work arrays…
      END SUBROUTINE SWCOMP
```

Fig. 4. Implementation of the pipelined parallel approach in the main computational routine of SWAN using OpenMP. OpenMP compiler directives begin with the !$OMP sentinel. The ellipses indicate actual code omitted for clarity.

in the main computational routine (which is called at each time step in a non-stationary application) in SWAN are shown in Fig. 4. The OpenMP directive lines begin with the "!$OMP" sentinel. After some pre-iteration setup and allocation of shared (visible to all threads) work arrays the parallel region begins, as indicated by the "!$OMP PARALLEL DEFAULT(SHARED)" directive, at which point a team of threads is started, the number of which is determined by the programming environment. The "DEFAULT(SHARED)" instruction indicates that by default variables in the routine will be shared among threads unless specified otherwise. Once the parallel region begins each thread allocates its own private (*i.e.*, visible only to itself) work arrays. The iterative solution with sweeps across the geographic grid is controlled by the ITER and SWPDIR loops. The pipelined parallel approach is applied using the "!$OMP DO SCHEDULE(STATIC,1)" directive on the IY loop. This directive instructs the compiler to statically assign iterations of the IY loop to threads in a round-robin fashion. The LLOCK array, which is reset prior to beginning each sweep, is a logical array that guarantees each thread cannot proceed to the next grid point until the dependency in the Y direction for that grid point has been satisfied. Dependencies in the X direction are satisfied by virtue of the row assignment to a thread. After the state at grid point (IX,IY) is computed by the routine SWOMPU, the thread processing that grid point signals it is finished by setting LLOCK(IX,IY) to FALSE. The "!$OMP FLUSH" directive instructs the compiler to ensure that each thread in the team has a consistent view of variables in memory. There is an implied barrier (thread synchronization) at the end of the IY loop to ensure each thread remains on the same sweep. The explicit barrier at the end of the SWPDIR loop ensures all threads are finished with the sweeps before checking convergence. Once the iterations are completed each thread deallocates its private work arrays. The "!$OMP END PARALLEL" directive signals the end of the parallel region, where the team of threads is terminated and only a single or master thread continues.

It should be noted that since none of the order of operations in SWAN have been changed the new parallel code matches "bit-for-bit" with the original sequential code. This has been verified with numerous tests on a variety of platforms. Because the OpenMP code modifications are in the form of compiler directives which appear syntactically as comments to a non-OpenMP compiler, the new parallel code can still be compiled with a non-OpenMP compiler to run on a single processor machine. No changes have been made to the user interface.

IV. RESULTS AND DISCUSSION

To understand the parallel performance of SWAN it is useful to examine two quantities known as *speedup* and *efficiency*. For a fixed problem size, the speedup on p processors over the single processor execution is defined as $S_p = t_1/t_p$, where t_1 is the sequential execution time and t_p is the execution time on p processors. Theoretically, the speedup can never exceed the number of processors. The efficiency, defined as $E_p = S_p/p$, is a measure of the fraction of time for which a processor is usefully employed. In an ideal parallel system and implementation, the speedup

is equal to p and the efficiency is equal to one. In practice, the speedup is less than p and efficiency is between zero and one, depending on the design of the parallel system and the parallel program. Exceptions to this occur for certain cache-bound applications which may exhibit superlinear speedup and efficiency greater than unity.

Given the nature of the pipelined parallel approach in SWAN we can derive analytical expressions for the ideal parallel performance. We assume, for a given application, that the time to compute the state at a geographic grid point is the same for all grid points. We also assume there is zero overhead due to thread management. Based on these assumptions and a simple counting process the expression for the ideal efficiency of a single sweep is derived as

$$E_{ideal}(x, y; p) = \frac{xy/p}{x\lceil y/p \rceil + (y-1) \bmod p}, \quad (4.1)$$

where x and y are the geographic grid dimensions and $\lceil\ \rceil$ is the ceiling function. Equation (4.1) is valid only when $p \leq \min(x, y)$. In the case of $p > \min(x, y)$ it is not possible to fully utilize all the processors; as a result the speedup becomes constant, *i.e.*, p is replaced with $\min(x, y)$ in the denominator on the right hand side of equation (4.1). The theoretical efficiency is plotted in Fig. 5a as a function of p for several geographic grid sizes. Peaks in efficiency, which become more pronounced as x is increased, occur where $\lceil y/p \rceil$ changes (this includes where p is a divisor of y). As y is increased the number of peaks also increases, but the relative size of the peaks is less significant. With the exception of a highly elongated grid, where the efficiency peaks again at $p = y$, the best performance range is where $p \leq y/2$. Note that when p is a divisor of both x and y then (4.1) becomes symmetric with respect to exchange of x and y. In general, if one chooses p such that p is a divisor of

y then the independent grid dimensions x and y become less important and the overall grid size, that is, $x \cdot y$, can be used when estimating the parallel performance of a selected application.

The measured efficiency of SWAN for a stationary application with nonuniform currents and an 80x60 spectral grid on a 32-processor IBM Regatta (1.3GHz Power4) is shown in Fig. 5b. The measured efficiency is per iteration (4 sweeps). We see that the symmetries and the peak positions of the measured efficiency agree with those of the ideal efficiency (Fig. 5a). However the measured efficiency is approximately 10 to 20% lower than the ideal efficiency. This is a manifestation of mainly thread overhead and remaining sequential portions of the program.

The parallel performance of SWAN is dependent on the amount of work available at each geographic grid point − that is, on the spectral dimensions and whether the submatrix is tridiagonal or banded. The measured efficiency of SWAN for several spectral grids with a fixed 64x64 geographic grid is shown in Fig. 6. Although the work per geographic grid point is higher for the 144x128 spectral grid, the best overall performance is observed for the 36x32 spectral grid. This is likely due to better cache utilization for the smaller spectral grid. In the case of the 18x16 spectral grid the amount of work available at each geographic grid point is very small. Therefore, a large fraction of time is spent in thread management and sequential regions of the program resulting in a severe drop in parallel performance. This point becomes significant when considering geographic grids that have a large percentage of dry (land) points where computations are skipped (although the points are still included in the geographic grid loops).

Once the spectral dimensions and submatrix type are fixed the parallel performance of SWAN is, in principle, dependent only on the geographic grid dimensions and the number of processors. However, even these three remaining parameters present a complexity that makes complete characterization of the parallel performance of SWAN impractical. We have chosen to limit the characterization to those points where p is a divisor of y (where the ideal efficiency is optimal). With this restriction we can reasonably consider only the overall geographic grid size when estimating the parallel performance. Fig. 7 shows the measured efficiency as a function of p for several geographic grids on the IBM Regatta. The total geographic grid size is expressed in units of 1024 using the symbol "K". The spectral grid is fixed at 36x32. As expected the efficiency increases as the
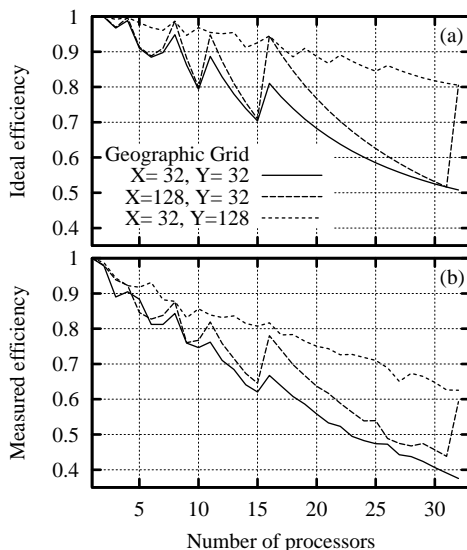
Fig. 5. Ideal (5a) and measured (5b) efficiency of SWAN for several geographic grid sizes (fixed spectral grid). The ideal efficiency is for a single sweep as computed from (4.1). The measured efficiency is per iteration (4 sweeps). Measurements were performed on a 32-processor IBM Regatta (1.3 GHz Power4).
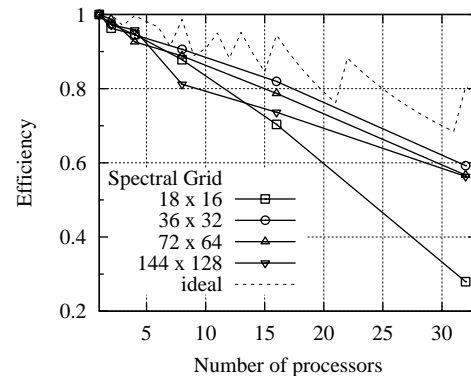
Fig. 6. Measured efficiency of SWAN for several spectral grid sizes with fixed depth and no currents. The geographic grid is fixed at 64x64.
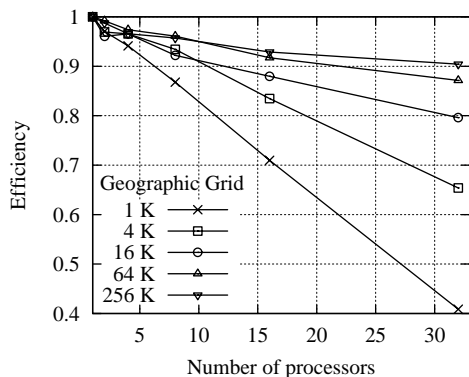
Fig. 7. Measured efficiency of SWAN for several geographic grid sizes with fixed depth and no currents. The spectral grid is fixed at 36x32. Total geographic grid sizes are listed in units of 1024 using the symbol "K".

geographic grid is increased. It appears that for large grids (512x512 and larger) the efficiency on 16 to 32 processors will not go much higher than 90%. Not only do the measurements in Fig. 8 provide a better understanding of trends, but they can also be used as a tool for estimating the performance when setting up a regional application of SWAN.

For an example application we consider a non-stationary simulation of Lake Michigan with 2km spatial resolution (126x248 geographic grid, 36x32 spectral grid). The simulation spans four days (November 8 - 12, 1995) with a time step of 0.1 hour, wind updates every hour, and data output every two hours. Fig. 8 provides snapshots of significant wave height and wave direction from November 10 and 12 of the Lake Michigan simulation. The simulations were performed on a 32-processor IBM Regatta (1.3GHz Power4). The sequential time for the four model day run was about 14.5 hours. However, on 18 processors the execution time was only about 1 hour, clearly demonstrating the benefit of parallel processing and the viability of SWAN for operational use.

Parallel performance measurements for the Lake Michigan case are shown in Fig. 9. With the exception of $p = 19$ the measurements were performed for p where the ideal efficiency has a local maximum. As discussed above, these are the points where $\lceil y/p \rceil$ changes, which includes those p where p is a divisor of y. The efficiency for the total run was obtained by measuring the time from start to finish. The efficiency per iteration was obtained by only measuring the time for the iteration loop which does not include file input and output that occurs between time steps. The data point at $p = 19$ is provided to show the effect of choosing p where the ideal efficiency has a local minimum. We see that the effect of this choice is small since for this geographic grid size the fluctuations in ideal efficiency are only about 2%.

From Fig. 8 we can see that although the total grid size is about 31K, the number of wet grid points is only about 13K (44%). The importance of considering the number of wet points when making an estimate of the parallel performance is demonstrated in Fig. 9 where the 16K efficiency per iteration (from Fig. 7) closely follows that of the Lake Michigan case. The efficiency for the total run is lower than the efficiency per iteration due to overhead from the time stepping and the sequential file input and output. We can make a reasonable estimate for the parallel efficiency of the
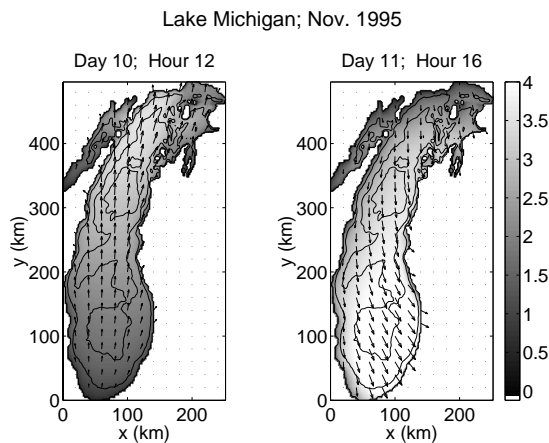
Lake Michigan; Nov. 1995

Fig. 8. Snapshots from a four-day simulation (November 8-12, 1995) of Lake Michigan using hourly wind input fields. Shading indicates the significant wave height and the arrows indicate the wave direction.

total run by measuring the execution times for various parts of the code during a short sequential run (say 2 model hours so as to include file input and output). Suppose, for a short sequential run, the total time is t_1 and the time spent in the sweep portion (where parallel execution can occur) of the code is t_{s1}. The fraction of time spent in sequential execution is then given by $f_s = (t_1 - t_{s1})/t_1$, which for the Lake Michigan case on the IBM Regatta turns out to be about 1%. Let $E_i(p)$ be the estimated efficiency per iteration (as obtained from Fig. 7), then the efficiency for the total run of the Lake Michigan case can be estimated as

$$E_{total}(p) = \left[\frac{(1 - f_s)}{E_i(p)} + pf_s \right]^{-1} . \qquad (4.2)$$

Equation (4.2) is plotted in Fig. 9 using 1% for the sequential fraction and the measured efficiency per iteration for a 16K grid. The agreement between the measured efficiency and the estimate for the total run is quite good. This establishes that, at least for this parallel machine, the measure for the
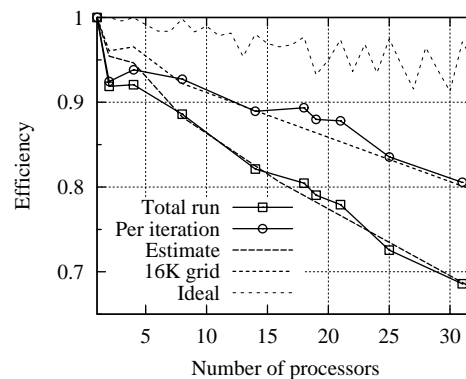

Fig. 9. Parallel performance of SWAN on the Lake Michigan case. The solid line with squares is the efficiency for the complete simulation. Efficiency per iteration, which does not include file input and output, is given by the solid line with circles. The 16K grid efficiency is taken from the results in Fig. 8. The estimated efficiency is obtained from the 16K efficiency and the estimated serial fraction. The ideal efficiency is from (4.1) with x=126 and y=248.

sequential fraction effectively describes most of the overhead that would arise in a parallel run. One can use the same procedure to estimate the parallel performance of SWAN for other regional applications.

V. SUMMARY

This paper has presented the development of a parallel version of the advanced third-generation wave model known as SWAN. Through careful analysis of the data and numerical algorithms the dependencies and potential for concurrent computation were determined. A pipelined parallel approach was chosen and implemented using OpenMP compiler directives which produce multithreaded code for shared-memory multiprocessor computers. The parallel implementation required no algorithmic changes and is 'bit-for-bit' compatible with the original sequential code. No change has been made to the user interface. An analytic expression was derived for the ideal parallel performance which displays the same features as the actual performance. Numerous performance measurements demonstrate the scalability of the parallel version of SWAN. The significant reduction in turn-around time for both stationary and non-stationary cases clearly establishes the viability of SWAN as an operational model. We conclude by pointing out that all code changes presented in this paper have been accepted by the SWAN developers to be included in the official release. Also, the parallel version of SWAN is presently undergoing transition into operational use for the Navy.

Acknowledgments

REFERENCES

[1] R.E. Jensen, P.A. Wittmann, and J.D. Dykes, "Global and regional wave modeling activities," *Oceanography*, vol. 15, no. 1, pp. 57-66, 2002.

[2] G.J. Komen, L. Cavaleri, M. Donelan, K. Hasselmann, S. Hasselmann, and P.A.E.M. Janssen, *Dynamics and Modelling of Ocean Waves*, Cambridge University Press, 1994.

[3] H.L. Tolman, "A third-generation model for wind waves on slowly varying, unsteady and inhomogeneous depths and currents," *J. Phys. Oceanogr.*, vol. 21, pp. 782-797, 1991.

[4] J.M. Smith, D.T. Resio, and A.K. Zundel, "STWAVE: steady-state spectral wave model, report 1 user's manual for STWAVE version 2.0," *Instr. Rep. CHL-99-1*, U.S. Army Corps of Engineers, Waterways Experiment Station, Vicksburg, MS, 1999.

[5] N. Booij, R.C. Ris, and L.H. Holthuijsen, "A third-generation wave model for coastal regions, 1. Model description and validation," *J. Geophys. Res.*, vol. 104, no. C4, pp. 7649-7666, April 1999.

[6] D.E.Culler, J.P. Singh, *Parallel Computer Architecture: A Hardware/Software Approach*, San Francisco, CA: Morgan Kaufmann, pp. 75-120, 1999.

[7] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*, San Francisco, CA: Morgan Kaufmann, 2001.