Benchmarking of the ADCIRC Hydrodynamic Model on Workstation Clusters

K.M. Dresback, R.L. Kolar, J.C. Dietrich, E.P. Gilje

Environmental Modeling/GIS Laboratory (EM/GIS)
School of Civil Engineering and Environmental Science
University of Oklahoma
Norman, OK 73019

Technical Report No. 03-01

(Keywords: shallow water modeling, GWC equation, finite elements)

Abstract

This report presents the results of a series of benchmarking tests conducted using the parallel version of the ADCIRC hydrodynamic model. In particular, these tests compared the performance of the original version of the ADCIRC model against that of a similar version that includes a predictor-corrector (implicit) time-stepping algorithm. As will be shown, the enhanced stability of this new algorithm allows simulations to be completed in less wall-clock time, without a decrease in scalability. We benchmarked these two versions of the model on two 16-processor workstation clusters: one consisting of Sun UltraSparc IIe processors, and another consisting of Intel Pentium III processors. Results indicate that the predictor-corrector algorithm provides the same enhanced stability in parallel that it does in serial, allowing for larger time steps and shorter wall-clock times. Results also indicate that the two platforms show similar performance, though the Intel platform is slightly better in some instances.

1. Background

Shallow water equations are used by researchers and engineers to model the hydrodynamic behavior of oceans, coastal areas, estuaties, lakes and impoundments (Kolar, et al., 1994). The finite element solutions of these equations have been improved by two equations: the wave continuity equation (WCE), introduced by Lynch and Gray (1979) to suppress the spurious oscillations inherent to the primitive equations without having to dampen the solution either numerically or artificially; and the generalized wave continuity equation (GWCE), introduced by Kinnmark (1986) to achieve a balance between the primitive and pure wave forms of the shallow water equations by replacing the bottom friction τ with a numerical parameter G. The finite element model used in this report, ADCIRC (an ADvanced three-dimensional CIRCulation model), was developed from the GWCE (Luettich et al. 1992; Westerink et al. 1994).

The ADCIRC model encounters stability problems in highly non-linear applications unless a severe Courant number (Cr) restriction is imposed. In deep water applications, a practical upper bound of the Courant number is 0.5 in order to mantain the stability of the model; an even tighter constraint must be imposed if the simulation includes barrier islands and constricted inlets (e.g. Cr < 0.1). In order to relax this restriction, an alternative time-marching algorithm procedure was implemented that treats some or all of the non-linear terms implicitly (Kolar, et al., 1998). This predictor-corrector algorithm has been shown to dramatically increase the maximum stable time step in a number of one- and two-dimensional applications (Dresback et al., 2001; Dresback et al., 2000). In this study, we examined the behavior of this algorithm in a parallel computing environment.

2. Methodology

The subject of this study was two-fold: (1) to compare the original and predictor-corrector versions of the ADCIRC (2DDI - depth integrated) model in a parallel computing environment; and (2) to compare the behavior of the ADCIRC model on two 16-processor clusters. The differences between these time-marching algorithms and computing clusters are described in this section.

2.1 *Time-Marching Algorithm: Original vs. Predictor-Corrector*

The background and effects of the predictor-corrector time-stepping algorithm have been discussed previously by Dresback et al. (2001). In short, the algorithm implicitly evaluates the non-linear terms in two stages. The predictor stage is equivalent to the existing semi-implicit algorithm, i.e., it evaluates the non-linear terms using the values from the present. Estimates of future values (called k*) obtained from the predictor step are combined with the already known present (k) and past (k-1) values to obtain the corrected values for the future time level (k+1). The corrector stage can be repeated as many times as necessary, but it has been shown that only one iteration is necessary to maximize stability (Dresback et al., 2001). Results from one- and two-dimensional simulations indicate that this new algorithm increases the maximum stable time step by at least 100 percent, the minimum required to overcome the additional overhead. Gains sometimes show as much as a ten-fold increase.

This study considers the implementation of this algorithm in a parallel computing environment. Tests were conducted on the two time-stepping algorithms as programmed in version 41.03 of the ADCIRC model: an original version, without any added features; and an altered version that includes the predictor-corrector algorithm. Both versions utilize the two-dimensional depth-averaged capabilities of the model. We then examined the effects of the new time-marching algorithm by comparing the results of the original parallel version of ADCIRC against those of the predictor-corrector version.

The size of the time step was also varied in some instances, as will be described in Section 3. In general, we examined two time-step relationships: one when the time step for the original and predictor-corrector simulations was set at an equal (but arbitrary) value; and another when the time steps for the original and predictor-corrector simulations were set at their maximum stable values. The former relationship allowed us to examine the additional computation time required by the corrector step in the new algorithm. The latter relationship allowed us to examine the increase in stability and efficiency afforded by the new algorithm.

2.2 *Workstation Clusters: Sun UltraSparc IIe vs. Intel Pentium III*

This study also considered the implementation of the ADCIRC model on two 16-processor computing clusters: one consisting of Sun UltraSparc IIe processors; and another consisting of Intel Pentium III processors. The characteristics of these clusters are shown in Table 1.

Table 1. Operating characteristics, software and cost of the two 16-processor computing clusters.

	Sun UltraSparc IIe	Intel Pentium III
Speed	500 MHz	1 GHz
Operating System	Solaris 8	Linux
Cache	256 KB	256 KB
Memory	128 MB	256 MB
Communication	100 Mb/s	100 Mb/s
Compiler	Sun Forte 6.0	NAG
MPI	Sun ClusterTools	MPIch
Total Cost	$19,300	$27,500

2.3 *Compiler Flags*

On the Sun platform, we did not encounter any significant problems while compiling the program on the Sun platform. To optimize performance on our system, we tested more than 40 combinations of compiler flags before selecting a set of flags that worked well. Tests were conducted on a small 10x10 quarter harbor domain, with a 50-second time step and a 30-day duration. The following combination of compiler flags offered the best performance:

-fast -xO4 -xdepend -fsimple=1 -f -dalign -xtarget=ultra -xarch=v8plusa

The -fast flag is a macro that expands to a set of options to optimize the code, and its components are listed in the manual pages for f90. The -xO4 flag is an optimization level in the -xO[n] family that includes all of the optimization from the three lower levels plus inlining of functions within the same file and aggressive global optimization. The -xdepend flag instructs the compiler to analyze data dependency within loops and perform loop restructuring, attempts cache blocking to reduce cache misses and loads, and assists the compiler when loop unrolling and applying other optimizations. The -fsimple=1 flag allows conservative simplifications of floating point arithmetic assumptions. The -f flag aligns all common blocks and all double- and quad-precision local data on 8-byte boundaries. The -dalign flag gets faster execution of double- and quad-precision computations and prevents memory errors and crashes. The -xtarget=ultra flag is a macro that expands into flags for -xarch, -xchip, and -xcache depending on the processor. The -xarch=v8plusa flag reuires UltraSPARC and allows VIS 1.0 instructions.

Without these compiler flags, the test simulation finished in slightly under five minutes, at 4:39. The addition of these compiler flags decreased the runtimes to 1:26, a decline of almost 70 percent. Thus, these optimization flags were used for all runs on the Sun platform.

To determine which set of compiler flags provided the optimal performance on the Intel platform, we compared the -O1, -O2, -O3, -O4, -Ounsafe, and -Oassumed flags. These tests were conducted on a different ADCIRC run from that used on the Sun platform. This Intel run, when compiled without any optimization flags, ran in a wall-clock time of 2:26. Tests on six combinations of the -O series of flags consistently ran in wall-clock times ranging from 0:58 to 1:11, a decline of more than 50 percent. All of the benchmarking on the Intel platform was done with the -O flag, which is also the same as the -O2 flag.

2.4 *Parameters of Benchmarking Studies*

The tests in this study can be broken into four parts. The first two parts consider an east coast domain and an ideal quarter annular harbor domain, in which the total numbers of global nodes were kept constant and were evenly distributed amongst a variable number of processors. The third part again considers the quarter annular harbor domain, but this time the number of nodes on each processor is kept at a constant value so that the number of global nodes varies with the number of processors. The fourth part also considers the quarter annular harbor domain, but the total number of nodes was varied over only one processor. These tests will be further described in this subsection, and the results of these tests will be shown in

Section 3.

The quarter annular harbor domain is a fictional grid, used in this study because its geometry allows for easy variations in grid resolution. It consists of a harbor shaped like a quarter circle, with an inner radius of 200,000 feet and an outer radius of 500,000 feet. The bathymetry varies quadratically, with a minimum depth of 10 feet at the inner land boundary and a maximum depth of 62.5 feet at the outer ocean boundary. The two radial sides are also land boundaries. An example of this grid is shown in Figure 1.

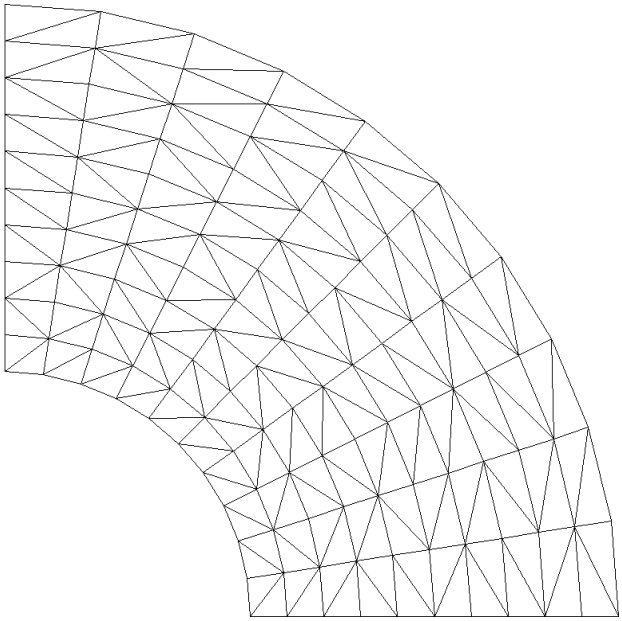

Figure 1. Example of quarter harbor domain. This is a relatively coarse example, with 121 nodes. The actual resolution of the grids in this study varied, as described in Section 2.

The east coast domain is a representation of the eastern coast of the United States, the western north Atlantic Ocean and the Gulf of Mexico. The domain is relatively coarse, with exactly 32,947 nodes. An example of this grid is shown in Figure 2.

Part 1. Benchmarking studies were first conducted on the east coast domain. Simulations were run with a four-day duration, two-day ramp, and a $G = \tau_0 = 0.005$ sec^{-1}. In addition, the time steps were varied according to two relations: one with the time step at 50 seconds, for both the original and predictor-corrector versions of the code; and one with the time step at its maximum stable value, 60 seconds for the original code and 515 seconds for the predictor-corrector. The first relation examines the effect of the additional computation time required for the predictor-corrector, while the second relation

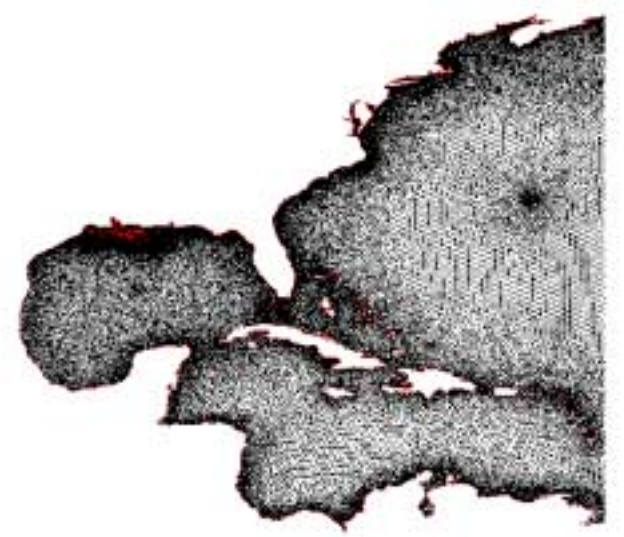

Figure 2. The east coast domain, which includes the eastern coast of the United States, the western north Atlantic Ocean and the Gulf of Mexico.

examines the effect of the additional stability afforded by the predictor-corrector.

Part 2. Similar benchmarking studies were also conducted on a quarter harbor domain consisting of 100,000 nodes. Simulations were run with a five-day duration, half-day ramp, and a $G = \tau_0 = 0.001$ sec^{-1}. The time steps were again varied according to two relations: one with the time step at 25 seconds, for both the original and predictor-corrector versions of the code; and one with the time step at its maximum stable value, 30 seconds for the original code and 180 seconds for the predictor-corrector. The first relation examines the effect of the additional computation time required for the predictor-corrector, while the second relation examines the effect of the additional stability afforded by the predictor-corrector.

Part 3. The ideal quarter harbor domain was also used to examine the timing behavior when the local problem size was held approximately constant on each processor. In these runs, each processor was responsible for the same average number of nodes; thus, the overall problem size increased as the number of processors increased. In this way, the cost of communication could be measured. The average number of local nodes was chosen to be 5,000, in order to keep the problem size non-trivial and the simulation length reasonable. Simulations were run with a five-day duration, half-day ramp, and a $G = \tau_0 = 0.001$ sec^{-1}. The time step was held at a constant value of 25 seconds.

Part 4. The fourth part of this study involved varying the problem size on just one processor. This is almost the exact opposite of what was done in the third part. Whereas in Part 3 the focus was almost exclusively on the cost of communication, here the focus is on the computational ability of just one processor. In particular, we looked for a relation between job size and the sizes of the cache and memory. The ideal quarter harbor domain was used again, and the problem size was varied by varying the number of nodes in the domain, from 50 to 560,000 nodes. Simulations were run with a five-day duration, half-day ramp, and a $G = \tau_0 = 0.001$ sec^{-1}. The time step was held at a constant value of 25 seconds.

3. Results

To establish the single-processor baseline, the simulations were conducted once in serial and fifteen times in parallel, yielding timing information for tests ranging from two to sixteen processors. On the Sun platform, serial runs were timed with the UNIX "time" command and parallel runs were timed with the Sun ClusterTools "ptime" command. On the Intel platform, the runs were timed with the LINUX "time" command. These commands return three different times: real, user, and system. The times presented in this paper are the real, or wall-clock, times.

The figures in this section follow a consistent color scheme: the original code is shown in blue, while the predictor-corrector code is shown in red; and the Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds. The details of each figure will be discussed in this section.

3.1 *East Coast Domain with Global Nodes Constant*

Figure 3 shows the wall-clock timing information for the east coast domain as a function of the number of parallel processors. All results are shown with the time step set at its maximum stable value, which is 50 seconds for the original code and 515 seconds for the predictor-corrector code. The stability provided by the predictor-corrector version allows this value to be more than eight times greater than it is for the original version. As a result, the predictor-corrector version runs in a much shorter time. For example, an eight-processor Sun simulation ran in 11 minutes with the original code and only four minutes with the predictor-corrector code, a decline of 63 percent. However, there does not appear to be any significant difference between the Sun and Intel platforms, particularly for the predictor-corrector simulations.

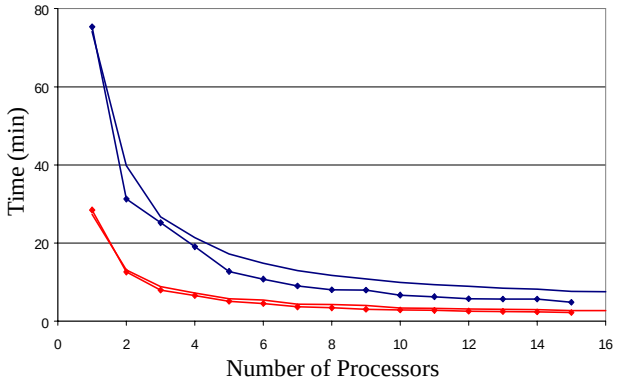

Figure 3. Timing information for the east coast domain. The original code is shown in blue, while the predictor-corrector code is shown in red; the Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds

Figure 4 shows the scaling information for the east coast domain. Again, the time step is set at its maximum stable value. The vertical axis, speed-up, is a function of how much faster the parallel version runs in comparison to a two-processor parallel version. A four-processor parallel simulation should theoretically run twice as fast as a similar two-processor parallel simulation. This theoretical speed-up line is shown in black on the figure.

The main difference is between the two platforms, not the two versions of the code. The Intel simulations show near theoretical speed-up through eight processors, but then appear to tail off as the number of processors is increased. On the other hand, the Sun simulations consistently show lesser speed-ups than the Intel simulations. Between the two versions of the ADCIRC code, though, there does not appear to be a significant difference. The original version and the predictor-corrector version show similar speed-up curves, depending on the platform.

3.2 *Quarter Harbor Domain with Global Nodes Constant*

Figure 5 shows the wall-clock timing information for the quarter harbor domain as a function of the number of parallel processors. All results are shown with the time step set at its maximum stable value, which is 30 seconds for the original code and 180 seconds for the

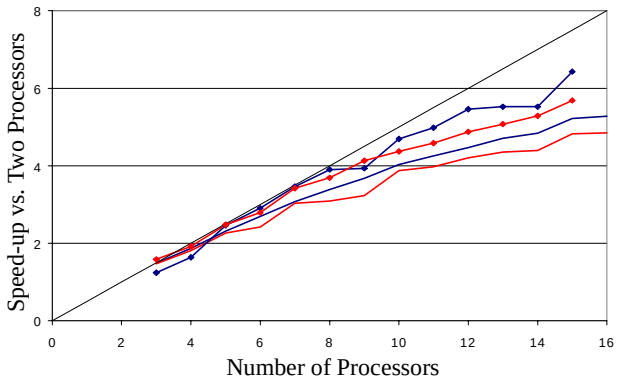

Figure 4. Scaling information for the east coast domain. The theoretical speed-up is shown with a smooth black line. The original code is shown in blue, while the predictor-corrector code is shown in red; the Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds.

predictor-corrector code. This domain shows a similar trend to that of the east coast domain. The enhanced stability provided by the predictor-corrector version allows the simulations to run in less wall-clock time. For example, an eight-processor Sun simulation ran in 64 minutes with the original code and only 36 minutes with the predictor-corrector code, a decline of 44 percent. This trend is the same for both platforms, as the original version takes longer on both the Sun and Intel systems.

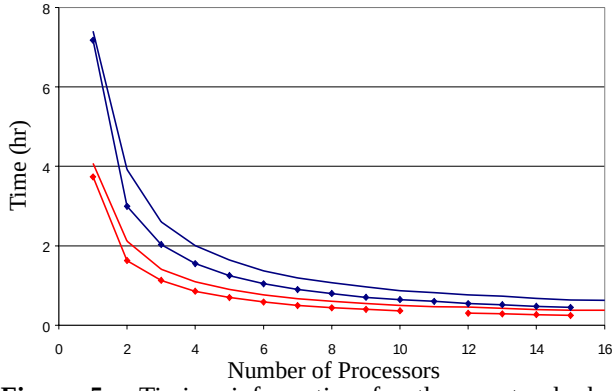

Figure 5. Timing information for the quarter harbor domain. The original code is shown in blue, while the predictor-corrector code is shown in red; the Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds.

Figure 6 shows the scaling information for the quarter harbor domain. The time step is set at its maximum stable value. The trend in this figure is also similar to that of the east coast domain in Figure 4. Both platforms provide near linear speed-up through about eight processors, and then the results begin to taper off. Also, the Intel platform provides slightly better results. The two versions of the ADCIRC code show no significant difference in speed-up, as their curves are similar for similar platforms.

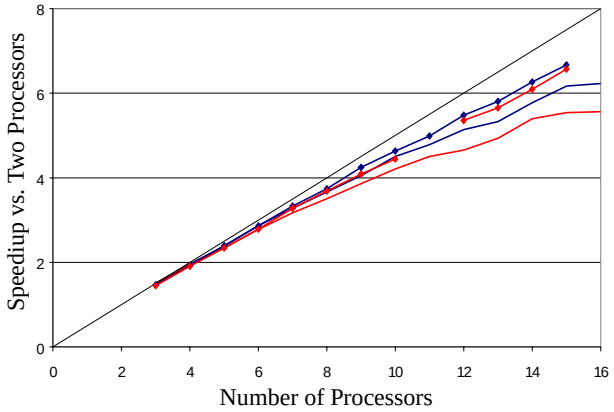

Figure 6. Scaling information for the quarter harbor domain. The theoretical speed-up is shown with a smooth black line. The original code is shown in blue, with the predictor-corrector code is shown in red; the Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds.

3.3 *Quarter Harbor Domain with Local Nodes Constant*

Figure 7 shows the wall-clock timing information for the quarter harbor domain as a function of the number of parallel processors. In this set of runs, the local problem size was held constant by varying the global problem size with the number of processors. Theoretically, these timing runs would show a horizontal line. This graph shows significant differences between both the versions of the code and the platforms on which they are run. The differences between the versions of the code can be explained by the constant time step. Because the predictor-corrector requires twice the computation per time step as the original version, it is reasonable for the overall simulation to also take twice as long. The differences between the Sun and Intel platforms will be discussed in Section 4. It is important to note that the Sun shows a significant increase in wall-clock time from one to two processors, while the Intel platform shows no such increase.

3.4 *Quarter Harbor Domain on One Processor*

Figure 8 shows the work per unit time for both platforms as a function of the number of processors. Work per unit time was calculated in two steps: first, we calculated the total memory used by all of the

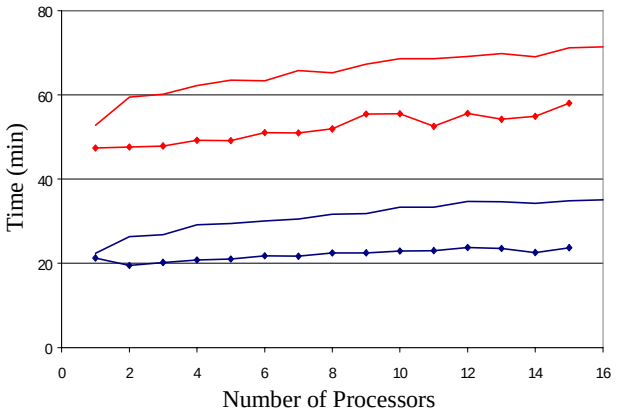

Figure 7. Timing information for the quarter harbor domain, with a constant local problem size. The original code is shown in blue, while the predictor-corrector code is shown in red; the Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds.

arrays for each grid; and second, we divided this total memory by the wall-clock time for each run. We only tested the original version of the code because we felt the predictor-corrector version would not show a significant difference. The amount of work would approximately double between the original and the predictor-corrector for the same time step, but the wall-clock time would also double, thus giving roughly the same information.

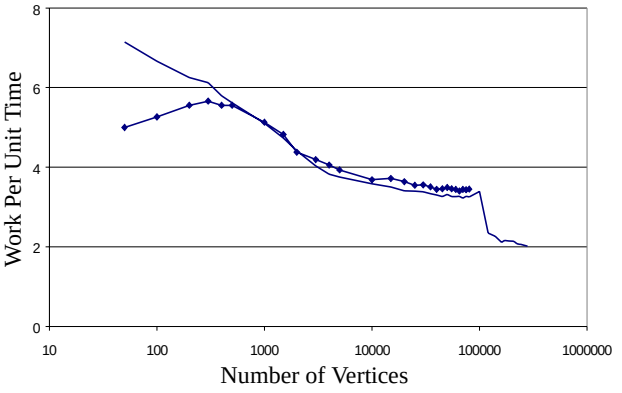

Figure 8. Timing information for the quarter harbor domain, showing work per unit time vs. problem size. The Sun simulations are shown with a smooth line, while the Intel simulations are shown with a line of diamonds.

The two operating platforms show almost identical curves, especially above 500 vertices. The only differences are in the very coarse grids, where the Sun platform continues to climb toward better work per unit time numbers, while the Intel platform tapers off from its peak. The Sun platform shows a drop-off in performance between 100,000 and 120,000 vertices. The Intel platform was not run on that fine of a grid because its batching system would not allow runs of those lengths of time. However, because both platforms have 256 KB caches, we believe the Intel platform would show similar behavior to the Sun platform for these fine grids.

4. Discussion

Although the results in Section 3 were shown in four parts, it is possible to lump together the discussion of the east coast and quarter harbor domains with constant global nodes, because of the similarities between their methods and results. Thus, discussion can focus on three main areas: global nodes constant, local nodes constant, and serial processor runs.

First, for the benchmarking studies that held the number of global nodes at a constant value, the results indicate that code version and operating platform have different effects in different areas. The predictor-corrector version of the code yields much shorter wall-clock times than does the original version, because its enhanced stability allows for a significantly higher time step. For the east coast, the difference in time step was a factor of eight; for the quarter harbor, the difference was a factor of six. Thus, even though the predictor-corrector requires roughly twice as much work per time step, its simulations are able to finish in shorter times.

The results also show that simulations on the Intel platform finish in less wall-clock time than those on the Sun platform, though the difference is not as pronounced as that between the two versions of the code. The differences between the two platforms can be explained by their operating characteristics: the Intel's 1 MHz speed and 256 MB of memory are both double the Sun's capabilities. That the Intel platform does not run twice as fast as the Sun platform is an indicator that the Intel platform has a bottleneck somewhere else that is slowing it down.

The scalability graphs for both domains indicate a minor difference between the two platforms. The Intel platform provides slightly better speed-up values than the Sun platform, though the difference is not significant through 16 processors.

Second, for the benchmarking studies that held the number of local nodes constant, both variables are significant. This time, the predictor-corrector version shows significantly longer wall-clock times than the original version, because both versions were run at

the same time step. The higher maximum stable time step afforded by the greater stability of the predictor-corrector version does not come into play (for this test), and thus is not able to counteract the extra time required by the corrector step. This trend holds for both operating platforms.

The difference between the platforms is their ability to handle parallel processing. The wall-clock times for their serial runs are very similar, but the times diverge for runs on two processors or more. The Intel platform shows only a mild increase in wall-clock time from one to two processors, and shows no significant change in wall-clock time as the number of processors is increased to sixteen. The Sun platform shows a significant increase in wall-clock time between one and two processors, of almost four minutes (17.5 percent) for the original version and almost eight minutes (16 percent) for the predictor-corrector. The rest of its parallel runs (for 3-16 processors), though, show no significant change in wall-clock time.

We speculate that this major difference between platforms is caused by how their communication is configured. The Intel platform is locally connected, with a master node that is the only node connected to the Internet at large and that assigns jobs to the others. On the other hand, each node of the Sun cluster is connected to the Internet backbone, so that users can log on and initiate jobs on any of the sixteen nodes. Thus, on the Intel cluster, communication between nodes is over a dedicated network that serves no other purpose, while on the Sun, communication between nodes is over a non-dedicated network that is used not only for this purpose but also for communication with the Internet at large. The increase in wall-clock time for the Sun platform from one to two processors is indicative of this cost of communication.

Finally, for the benchmarking studies on only one processor, the results indicate that there is no major difference between the computing performances of the two platforms. Both show a gradual decrease in performance before declining significantly when the problem size reaches 100,000 vertices. Thus, for processors with a 256 KB L2 cache, it is recommended that the sizes of parallel runs be limited so that any one processor is not responsible for more than 100,000 vertices.

5. Conclusions

The primary objectives of this report were to examine the results of a series of benchmarking tests conducted using the parallel version of the ADCIRC hydrodynamic model. These tests compared the performance of the original version of the ADCIRC model against that of a similar version that includes a predictor-corrector time-stepping algorithm. We also compared the performance of two parallel computing clusters: one consisting of Sun UltraSparc IIe processors, and another consisting of Intel Pentium III processors.

Significant conclusions that we can draw from this study are listed below.

- The enhanced stability of the predictor-corrector time-stepping algorithm, already verified in serial, is also evident in parallel. The algorithm allows for a higher maximum stable time step, which in turn allows for shorter wall-clock times.
- The Intel platform is slightly faster than the Sun platform on problems where the number of global nodes is held constant and distributed over a variable number of parallel processors.
- Both platforms show similar speed-up values, though the Intel does slightly outperform the Sun in some instances.
- The Intel platform offers better communication between parallel processing nodes than the Sun platform. While there is a significant difference in wall-clock times between one and two processors on the Sun platform, the Intel platform shows only a slight increase.
- Both processors show a similar computing performance on one processor. For a platform with a 256 KB cache, local problem size should be limited to 100,000 vertices.

6. Acknowledgments

Financial support for this research was provided in part by the National Science Foundation under the contracts ACI-9623592 and EEC-9912319 and by the U.S. Department of Defense under contract ONR-N000140210651. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation. The

authors would also like to thank Henry Neeman, director of the O.U. Supercomputing Center for Education and Research (OSCER).

7. References

Dresback KM, Kolar RL. An implicit time-marching algorithm for 2-D GWC shallow water models. In *CMWR XIII: Computational Methods in Water Resources*, vol. 2, Bentley LR, Sykes JF, Brebbia CA, Gray WG, Pinder GF (eds). A.A. Balkema: Rotterdam, 2000: 913-920.

Dresback KM, Kolar RL. An implicit time-marching algorithm for shallow water models based on the generalized wave continuity equation. *International Journal for Numerical Methods in Fluids* 2001; **36**: 925-945.

Kinnmark IPE. The shallow water wave equations: formulations, analysis and application. In *Lecture Notes in Engineering*, vol. 15, Brebbia CA, Orsszag SA (eds). Springer-Verlag: Berlin, 1986; 187.

Kolar RL, Gray WG, Westerink JJ, Luettich RA. Shallow water modeling in spherical coordinates: equation formulation, numerical implementation and application. *Journal of Hydraulic Research* 1994; **32**(1): 3-24.

Kolar RL, Looper JP, Westerink JJ, Gray WG. An improved time marching algorithm for GWC shallow water models. In *CMWR XII: Computational Methods in Surface Flow and Transport Problems*, vol. 2, Burganos VNI, Karatzas GP, Payatakes AC, Brebbia CA, Gray WG, Pinder GF (eds). Computational Mechanics Publications: Southampton, 1998: 379-385.

Luettich RA, Westerink JJ, Scheffner NW. ADCIRC: an advanced three-dimensional circulation model for shelves, coasts and estuaries. Report 1: theory and methodology of ADCIRC-2DDI and ADCIRC-3DL. Technical Report DRP-92-6, Department of the Army, USACE, Washington, DC, 1992.

Lynch DR, Gray WG. A wave equation model for finite element tidal computations. *Computers and Fluids* 1979; **7**(3): 207-228.

Westerink JJ, Luettich RA, Blain CA, Scheffner NW. ADCIRC: an advanced three-dimensional circulation model for shelves, coasts and estuaries. Report 2: Users Manual for ADCIRC-2DDI, Department of the Army, USA