A Direct Out-of-Core Solver for CGWAVE on the Cray X1

Fred T. Tracy, Thomas C. Oppe, and Zeki Demirbilek
USACE Engineering Research and Development Center (ERDC), Vicksburg, MS
{Fred.T.Tracy, Thomas.C.Oppe, Zeki.Demirbilek}@erdc.usace.army.mil

Abstract

CGWAVE is a general-purpose, state-of-the-art wave-prediction model. It is applicable to the estimation of wave fields in harbors, open coastal regions, coastal inlets, around islands, and around fixed or floating structures. CGWAVE generates a system of simultaneous linear complex equations that are difficult to solve. Some iterative solvers do not converge at all, while others struggle to converge. A direct, banded, out-of-core solver that utilized reordering of the nodes to reduce the size of the bandwidth was written. The solver was then optimized for the US Army Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC) Cray X1. This paper will describe the details of how this work was performed and give performance results on the ERDC MSRC Cray X1.

1. CGWAVE

CGWAVE[1, 2] is a general-purpose, state-of-the-art wave-prediction model. It is applicable to the estimation of wave fields in harbors, open coastal regions, coastal inlets, around islands, and around fixed or floating structures. Thus, the results from CGWAVE can have significant military applications. Figure 1 shows a generic computational domain. Both monochromatic and spectral waves can be simulated with the CGWAVE model. CGWAVE is a finite element model that is interfaced to the Corps of Engineers' Surface-Water Modeling System (SMS)[4] for graphics and efficient creation of finite element mesh generation and other input data.

CGWAVE was used to model harbors of military interest in various parts of the world to create a database for use by Department of Defense (DoD) organizations. This information is vital for planning and execution of military operations, training, and exercises.

1.1. Suitability.

CGWAVE is particularly suited for performing wave simulations in regions with arbitrarily shaped (man-made or natural) boundaries and arbitrary depth variations. Intrinsic limitations do not exist on the shape of the domain, the angle of wave incidence, or the degree and direction of wave reflection and scattering that can be modeled. CGWAVE is the DoD's harbor-wave simulation model. It is used (a) in the planning and execution of operations in ports and harbors and (b) design/modification of commercial ports, marinas, and yacht basins. Engineers conducting studies on navigation, channel deepening, and fluid-structure interaction problems can make use of CGWAVE, also.

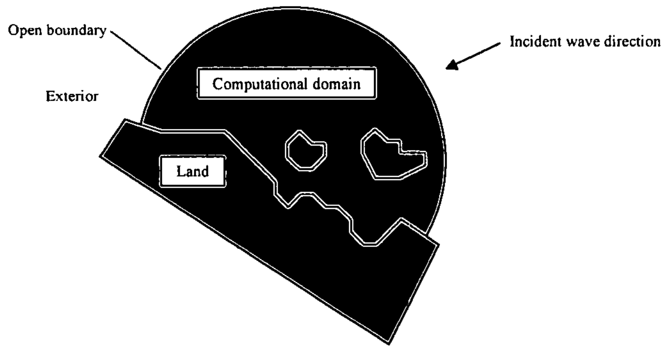

Figure 1. Computational domain

1.2. Governing Equations.

CGWAVE is based on the solution to the elliptic mild-slope equation (MSE) for modeling surface-gravity waves in coastal areas. The MSE represents integration over water columns of the three-dimensional Laplace's equation used in the classical potential wave theory. The governing equation is as follows:

$$\nabla\cdot\left(CC_g\hat{\eta}\right)+\frac{C}{C_g}\sigma^2\hat{\eta}=0$$

$$C=\frac{\sigma}{k}$$

$$C_g=\frac{\partial\sigma}{\partial k}=nC \qquad (1)$$

$$n=\frac{1}{2}\left[1+\frac{2kd}{\sinh(2kd)}\right]$$

$$\sigma^2=gk\tanh(kd)$$

where $\hat{\eta}$ is the complex surface elevation function from which the wave height can be determined, σ is the wave frequency under consideration (radians/sec), C is the phase velocity, C_g is the group velocity, k is the wave number, and d is the local water depth.

CGWAVE provides an estimate of the spatial variation in the wave field from the time incident waves enter the model domain in the deep ocean to the time they move through inlets/entrances and start impacting various maritime activities inside ports, harbors, and estuaries. The wave-field data model provides (a) wave height, direction, and speed, (b) pressure, and (c) wave- radiation stresses.

1.3 Linear Solvers.

Whether finite differences or finite elements are used for the discretization of the governing equations and boundary conditions of CGWAVE, the numerical treatment leads to a system of simultaneous, linear complex equations,

$$\mathbf{Ax}=\mathbf{b} \qquad (2)$$

that are difficult to solve. Here, **A** is the nonsymmetric, complex matrix of coefficients, **x** is the vector of unknown complex potentials, and **b** is a vector of known complex terms. The matrix **A** is usually extremely large, and earlier attempts to solve Eq. 2 by Gaussian elimination had limited success because of the enormous memory and compute time required when the number of wavelengths in the domain is large. This difficulty can be partially alleviated by using frontal solvers. In recent years, most solutions have been based on the conjugate gradient (CG) method that guarantees convergence under the proper conditions. CG has been extremely robust in a wide variety of applications involving both finite differences and finite elements for several kinds of boundary conditions. Some researchers have used BiCG-Stab[5], which is efficient when it works but does not guarantee convergence for the governing equation used in CGWAVE. Others have explored the use of the Generalized Minimum Residual (GMRES) method and have reported greater efficiency with finite difference methods than finite element methods. With the weighted residual continuous Galerkin finite element formulation used in CGWAVE, the GMRES method fails to converge. Lastly, others have tried the multigrid method and found that it was best suited for rectangular finite-difference discretizations. In recent years, Dr. Thomas C. Oppe, ERDC Major Shared Resource Center (MSRC), has successfully solved the CGWAVE equations using a direct solver and has experimented with different types of preconditioners for iterative solvers. Unfortunately, in model simulations performed on PCs, memory requirements prevent the use of many solvers that can be used for medium- and large- domain applications using high performance computing (HPC).

To give the user and developer consistent software from PCs to HPC platforms, a direct out-of-core solver with partial pivoting for complex numbers was implemented as an option to the CGWAVE user. Because of the multistreaming and vectorizing capability of the Cray X1, this old approach offers an intriguing new potential. This is especially true since for all problems that were tested, a correct solution was achieved using the developed out-of-core solver.

2. Out-of-Core Solver with Partial Pivoting

A few details of the out-of-core solver used in conjunction with node reordering (Figure 2) will now be presented.

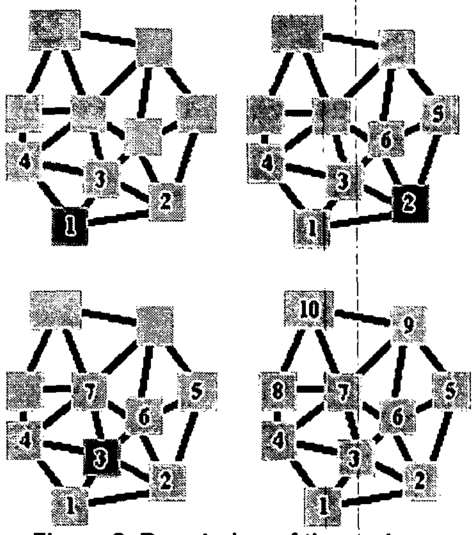

Figure 2. Reordering of the nodes

2.1. Reordering Nodes to Reduce the Bandwidth.

A banded system of equations guarantees that beyond a certain number of columns on either side of each main diagonal term, the elements in the **A** matrix are zero. So the half bandwidth h and bandwidth b are computed by

$$h = \max_{i,j,k} \left| n_{ij} - n_{ik} \right| \qquad b = 2h + 1 \tag{3}$$

Where n_{ij} is the jth node of element i, and n_{ik} is the kth node of element i. Figure 2 shows a simple algorithm to reorder the nodes to reduce the bandwidth. The new node 1 is placed anywhere on the mesh. Then, all nodes connected to the new node 1 are numbered consecutively. Next, all nodes connected to the new node 2 are numbered consecutively. This process is continued until all nodes have a new number.

To try every node in the mesh as a candidate for the new node 1 is prohibitive in computation time. So the strategy is to either use a more sophisticated algorithm or consider an arbitrary number, say 100, of different starting points equally spread among the given mesh. Ideas of reordering to reduce fill as provided in METIS[3] can also be pursued. The partial pivoting could cause an extra challenge because it is not known which of the rows will have the largest term for the partial-pivoting step before the computation starts.

2.2. Memory Allocation and Disc Storage.

Figure 3 shows how the elements of the **A** matrix are placed in memory for a problem with a bandwidth of 5 before and after pivoting between the first and third rows. Also, the number of the columns of a block is chosen to be the size of the bandwidth. Therefore, 3h + 1 rows and two blocks are all that are needed in memory at any one time. Figure 4 shows the main kernel of the forward part of the Gaussian elimination. The code for reading and writing of blocks and the partial pivoting is omitted.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{13} & a_{24} & a_{35} & a_{46} & a_{57} \\ 0 & a_{12} & a_{23} & a_{34} & a_{45} & a_{546} & a_{67} \cdots\cdots \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} & a_{77} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} \\ a_{31} & a_{42} & a_{53} & a_{64} & a_{75} & a_{86} & a_{97} \end{bmatrix}$$

a. Original storage

$$\begin{bmatrix} 0 & 0 & 0 & 0 & a_{35} & 0 & 0 \\ 0 & 0 & 0 & a_{34} & 0 & 0 & 0 \\ 0 & 0 & a_{33} & a_{24} & 0 & a_{46} & a_{57} \\ 0 & a_{32} & a_{23} & 0 & a_{45} & a_{546} & a_{67} \cdots\cdots \\ a_{31} & a_{22} & a_{13} & a_{44} & a_{55} & a_{66} & a_{77} \\ a_{21} & a_{12} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} \\ a_{11} & a_{42} & a_{53} & a_{64} & a_{75} & a_{86} & a_{97} \end{bmatrix}$$

b. After pivoting

Figure 3. Storage of the banded system of equations

```
do m = 1, nblocks
  do kk = kst, kend
    do jj = jst, jend
      j = jj - kst + 1
      rfact = row(jj - jst + 2) * afact
      if (rfact .ne. czero) then
        mij = jst - jj + nband
        mik = nband + 1
        do ii = ist, iend
          aa(mij, j) = aa(mij, j) - aa(mik, k) * rfact
          mij = mij + 1
          mik = mik + 1
        end do
      end if
    end do
    mik = nband + 1
    if (bfact .ne. czero) then
      do ii = ist, iend
        phi(ii) = phi(ii) - aa(mik, k) * bfact
        mik = mik + 1
      end do
    end if
  end do
end do
```

Figure 4. Main kernel of the forward part of the Gaussian elimination

3. Cray X1 Optimization

Vectorizing and multistreaming programs for the Cray X1 is absolutely essential for peak performance. One tool for accomplishing this task is compiler directives. Two examples encountered in the optimization of this out-of-core solver will now be shown. The first example (Figure 5) shows FORTRAN code for shuffling the nodal phi values with regard to the new node numbers nos. Because the compiler does not know that nos does not contain a given number only once, it cannot automatically multistream. However, after applying the directive, !dir$ concurrent, the loop will successfully multistream. The meanings of the compiler notations are as follows:

M-Multistreamed, V-Vectorized, r-unrolled, i-interchange

Before directive

```
r V--<>  temp(nos(1:nn)) = phi(1:nn)
V M--<>  phi(1:nn) = temp(1:nn)
```

After directive

```
!dir$ concurrent
r V M--<>  temp(nos(1:nn)) = phi(1:nn)
V M----<>  phi(1:nn) = temp(1:nn)
```

Figure 5. Shuffle node data

The second example is given in Figures 6–8 and represents the storing of the original sparse format **A** matrix into the blocks for writing to disk. Because of all the switching between old node numbers and new node numbers, the compiler cannot automatically multistream these do loops. However, aa(i, j) is not accessed more than once; so the code should multistream with compiler directives. Figure 7 shows the result of using the !dir$ concurrent directive. This is better, but the best notations from the compiler are the MV combinations. Figure 8 shows that this combination is achieved by using the Cray stream directive, !csd$ parallel do. The concept is very similar to that used in OpenMP.

```
1 2---< do jj = 1, nm
1 2 V-<   do ii = 1, nn
1 2 V       nold = nns(ii)
1 2 V       jold = id(nold, jj)
1 2 V       if (jold .ne. 0) then
1 2 V         node = nos(jold)
1 2 V         if ((node .ge. jst) .and.
(node .le. jend)) then
1 2 V           i = ii - node + nband
1 2 V           j = node - jst + 1
1 2 V           aa(i, j) = a(nold, jj)
1 2 V         end if
1 2 V       end if
1 2 V->   end do
1 2---> end do
```

Figure 6. Store into banded matrix without directives

```
1         !dir$ concurrent
1 iV---< do jj = 1, nm
1 iV 3-<   do ii = 1, nn
1 iV 3       nold = nns(ii)
1 iV 3       jold = id(nold, jj)
1 iV 3       if (jold .ne. 0) then
1 iV 3         node = nos(jold)
1 iV 3         if ((node .ge. jst) .and.
node .le. jend)) then
1 iV 3           i = ii - node + nband
1 iV 3           j = node - jst + 1
1 iV 3           aa(i, j) = a(nold, jj)
1 iV 3         end if
1 iV 3       end if
1 iV 3->   end do
1 iV---> end do
```

Figure 7. Store into banded matrix with !dir$ concurrent directive

```
1         !csd$ parallel do private (jj, ii,
1         !csd$& nold, jold, node, i, j)
1 M----< do jj = 1, nm
1 M MV-<   do ii = 1, nn
1 M MV       nold = nns(ii)
1 M MV       jold = id(nold, jj)
1 M MV       if (jold .ne. 0). then
1 M MV         node = nos(jold)
1 M MV         if ((node .ge. jst) .and.
(node .le. jend)) then
1 M MV           i = ii - node + nband
1 M MV           j = node - jst + 1
1 M MV           aa(i, j) = a(nold, jj)
1 M MV         end if
1 M MV       end if
1 M MV->   end do
1 M----> end do
1         !csd$ end parallel do
```

Figure 8. Store into banded matrix with !csd$ parallel do directive

3.1. Performance Results.

Table 1 shows timing results for three problems of various sizes. The following observations can be made: (a) the larger the bandwidth, the more increased the compute time; (b) the combination of vectorizing and multistreaming does an excellent job in reducing the time for the computations; and (c) the IO for the X1 is now the dominant computation. Using the FFIO capability, the X1 can reduce the IO time further, and this is a topic for further study.

Acknowledgment

This work was supported in part by a grant of computer time from the DoD High Performance

Computing Modernization Program at the ERDC MSRC, Information Technology Laboratory, Vicksburg, MS.

References

1. Demirbilek, Z., CGWAVE (computer program), http://chl.wes.army.mil/research/wave/wavesprg/numeric/wentrances/cgwave.htp, US Army Engineer Research and Development Center (ERDC), Vicksburg, MS, 2004.

2. Demirbilek, Z. and V. Panchang, "CGWAVE: A Coastal Surface Water Wave Model of the Mild Slope Equation." Technical report, US Army Engineer Research and Development Center (ERDC), Vicksburg, MS, 1998.

3. Karypis, G., METIS (computer program library), http://www.users.cs.umn.edu/~karypis/metis/metis.html, University of Minnesota, MN, 2004.

4. SMS Team, SMS (computer program), http://chl.erdc.usace.army.mil/CHL.aspx?p=s&a=Software;4, US Army Engineer Research and Development Center (ERDC), Vicksburg, MS, 2004.

5. Tang, J., Y. Shen, Y. Zheing, and D. Qiu, "An Efficient and Flexible Computational Model for Solving the Mild Slope Equation." *Coastal Engineering, Elsevier*, 2004, pp. 143–154.