A practical parallel retrofit of a 3-dimensional surface water model

M.J. Eppstein,[a] J.P. Laible[b]
[a] *Department of Computer Science and Electrical Engineering,* [b] *Department of Civil and Environmental Engineering, University of Vermont, Burlington, Vermont 05405, USA*

1 Abstract

Laible [2] has implemented a sequential Fortran-77 code for a 3-dimensional surface water flow model. The model uses the Galerkin Finite Element Method (FEM) with triangular elements and linear basis functions to solve the vertically integrated wave formulation of the shallow water equations for surface water elevations in the 2-d horizontal domain. The full 3-dimensional velocity field is subsequently solved for by the Galerkin FEM along the 3rd (vertical) dimension using 1-dimensional linear elements at each of the nodes in the 2-D horizontal mesh. The inclusion of a baroclinic term enables simulation of the internal seiche that develops in lakes under stratified conditions.

In this study, the existing code was parallelized using PVM Version 3.2.6 [5, 6]. A major goal of this parallel retrofit was to explore a heterogeneous programming paradigm that could be quickly and easily applied to complex existing codes, while still yielding useful speedups. Suboptimal parallelism, including a serial bottleneck and a high frequency of message passing, was deliberately accepted in order to minimize changes to existing code. While more efficient parallel algorithms are appropriate when writing new codes, they are seldom applied to the vast quantity of existing codes due to the need to rewrite major portions of the code. We demonstrate that useful speedups can be achieved with minimal programming effort, if optimal parallelism is sacrificed for the sake of simplicity of the port.

2 Introduction

Lakes, oceans, and estuaries are often modeled by a special form of the general Navier-Stokes equations of fluid dynamics. The special form is obtained by making several simplifying assumptions that realistically represent the true nature of the physical setting. The most fundamental of these is that the body of water is many times greater in horizontal extent than the depth. When this is the case, the general three dimensional equations can be represented by differential equations that can be solved sequentially for the unknown field variables. This is accomplished by discretizing the domain into a horizontal finite element mesh in the X-Y plane, with the Z-dimension further discretized into a 1-D "stretched" grid of a fixed number of vertical nodes at each node in the X-Y grid. The equations that emerge ($\Longrightarrow$ associated matrix approximations) are as follows.

The wave equation (1). This equation is used to solve for the surface elevation (ζ) and typically involves a finite element discretization over the horizontal (X-Y plane) surface of the water body.

$$\begin{gathered}\frac{\partial^2\zeta}{\partial t^2} + \tau_o\frac{\partial\zeta}{\partial t} - \vec{\nabla}_{xy}\cdot\left(g\frac{\rho_\zeta}{\bar{\rho}}H\vec{\nabla}_{xy}\zeta + \vec{M}_v - \tau_o H\vec{V}\right) = 0\\ \Longrightarrow\ [\mathbf{K}_\zeta]\{\zeta\} = \{\mathbf{P}_\zeta\}\end{gathered} \tag{1}$$

$$\begin{gathered}\tau_o = penalty\ parameter\ ,\ g = gravity\ ,\ H = total\ depth = h+\zeta\\ t = time\ ,\ h = mwl\ depth\ ,\ \rho_\zeta = surface\ density\ ,\ \bar{\rho} = \frac{1}{H}\int_{-h}^{\zeta}\rho\,dz\\ \vec{V} = \left\{\begin{array}{c}U\\V\end{array}\right\}\ ,\ U = \frac{1}{H}\int_{-h}^{\zeta}u\,dz\ ,\ V = \frac{1}{H}\int_{-h}^{\zeta}u\,dz\ ,\ \vec{\nabla}_{xy} = \left(\frac{\partial}{\partial x}\frac{\partial}{\partial y}\right)\end{gathered}$$

The vertically dependent momentum (2) *and continuity* (3) *equations.* The momentum equations are used to solve for the horizontal velocities u, v in the x(east) and y(north) directions respectively. Subsequently, the continuity equation is used to solve for the vertical velocities w. These unknowns are defined on a vertical string of nodes in the Z-dimension located at each of the horizontal nodes in the X-Y plane. At any horizontal location the vertically averaged velocities (U, V) can then be found by a vertical averaging of u and v.

$$\frac{\partial\vec{V}_c}{\partial t} + \lambda\vec{V}_c - \frac{\partial}{\partial z}\left(\epsilon\frac{\partial\vec{V}_c}{\partial z}\right) + \vec{f}_c = 0 \Longrightarrow [\mathbf{K}_v]\left\{\begin{array}{c}u\\v\end{array}\right\} = \{\mathbf{P}_v\} \tag{2}$$

$$\begin{gathered}\left(\epsilon\frac{\partial\vec{V}_c}{\partial z}\right)_{surface} = \vec{\tau}_s\ ,\ \left(\epsilon\frac{\partial\vec{V}_c}{\partial z}\right)_{bottom} = \vec{\tau}_b\\ \vec{V}_c =\ complex\ velocity\ = u+iv\ ,\ i = \sqrt{-1}\\ \lambda = f\ i\ ,\ f = Coriolis\ Parameter\\ \epsilon =\ vertical\ eddy\ vis\cos ity\ ,\ t = time\\ \vec{\tau}_s = \tau_{sx} + i\tau_{sy}\ ,\ \vec{\tau}_s = \tau_{bx} + i\tau_{by}\end{gathered}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \implies [\mathbf{K}_w]\{w\} = \{\mathbf{P}_w\} \tag{3}$$

The density transport equation (4) When the water body has a variable density structure, this equation must be solved for the density structure (ρ) at any horizontal location and at each vertical node.

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y} + w\frac{\partial \rho}{\partial z} = 0 \implies [\mathbf{K}_\rho]\{\rho\} = \{\mathbf{P}\rho\} \tag{4}$$

The terms $\vec{M}_v$ and $\vec{f}_c$ involve functions, derivatives and integrals of the unknowns u, v, w, ρ, U, V and ζ. All derivatives and integrals are evaluated numerically using the values at the discrete nodes. For a more complete discussion of the model see [2].

The bulk of the computational effort is expended in the formulation and solution of the equations for the Z-dimension (equations (2),(3),and (4)). Given that these are completely uncoupled in the horizontal dimension, a parallel implementation was attractive.

The serial version of the code was previously verified on standard test problems and applied to a Lake Champlain model [2].

3 Parallel Implementation using PVM

A combinaton of programming paradigms was employed in the parallel implementation of the model. Functional parallelization was used to overlap output with computation, the Single Program Multiple Data (SPMD) approach was used to parallelize the bulk of the computations in the simulation, and serial code was retained for the solution of the highly coupled horizontal system of equations. In some cases, the same computations are performed redundantly by all SPMD processes. The overriding goal of the parallelization was to achieve as much parallelization as possible without rewriting any of the existing code. PVM 3.2.6 was used to provide explicit message passing between processes in order to maintain sequential consistency [3] of the code.

In the PVM version, the user initiates execution of one copy of the SPMD process, known as the "parent" process. The parent then spawns multiple copies of the same process and each process is assigned a mutually exclusive set of contiguous nodes in the domain, known as a "subdomain". The code for each SPMD process is a (slightly modified) copy of the original serial code. The process with the most nodes in its subdomain is designated as the "solver" process. One new subroutine was written to create the processes and send them a startup message, and another new subroutine was created for the determination of subdomains by each process. Subdomain boundaries are specified via a short data file constructed by the user and need not be equally sized, so that heterogeneity in computing components may be used to advantage. For the determination of speedups, subdomains were kept roughly equal in size and were allocated to homogeneous processors.

This implementation requires that nodes and elements in the horizontal mesh be renumbered so that subdomains have contiguously numbered nodes and elements. For greatest efficiency, numbering should be transverse to the longest axis of the domain, as this allows subdomains to be cut along the shortest dimension of the domain thereby minimizing the number of nodes adjacent to each subdomain (and hence minimizing the size of the messages that must be passed between adjacent subdomains). No attempt has been made at this point to automate determination of subdomain boundaries, although this could be done if desired.

Each parallel SPMD process executes the initialization segment of the code (i.e. before the timestep loop) *exactly* as was done in the serial code. The data structures are constructed by each SPMD process for the *entire* domain, with each process reading a copy of the data files from the local disk of the processor on which it is running. This approach ensures that each process starts out with the correct data and requires no modifications to the initialization code, while minimizing disk contention during file access when running in a distributed disk environment, such as on a cluster of workstations. Although no speedup is possible during this portion of the program, the one-time cost associated with it is inconsequential in comparison to the time requirements of the timestep loop when executed over any realistic number of timesteps.

The real opportunities for speedup lie in the parallelization of the code within the timestep loop. In the existing 3-D surface water model, the bulk of the computational effort is expended in the calculation of the 3-D velocity and density fields. The equations to solve for the velocities and densities at each vertical string of nodes in the third dimension are highly nonlinear. The required coefficient matrices $[K_v]$, $[K_w]$, and $[K_\rho]$ must be recalculated each timestep. However, by virtue of the computational algorithm, these equations are completely uncoupled in the horizontal mesh, enabling maximal parallelism between each vertical string of nodes in the 2-D mesh.

The highly coupled nature of the data access required in the 2-D direct solver for surface elevations makes it difficult to parallelize this step without completely replacing the direct solver with a domain-decomposition algorithm (such as described by [4]). Domain-decomposition solvers come with computational costs of their own due to the iterative nature, whereas the symmetric, positive definate nature of the coefficient matrix $[K_\zeta]$ makes direct (non-iterative) solution of the equations relatively fast.

In this PVM implementation, a practical approach to parallelism was taken, whereby the relatively fast 2-D horizontal direct solver for surface elevations was allowed to remain as a serial bottleneck, performed by the

SPMD process with the largest subdomain. All nonlinear terms are embedded in the right-hand side $\{P_\zeta\}$, so that the coefficient matrix $[K_\zeta]$ contains only linear terms and is calculated only once (before the timestep loop). Hence, only one vector $\{P_\zeta\}$ must be gathered from the other SPMD processes, and only one vector $\{\zeta\}$ must be scattered back to the other SPMD processes, before and after the serial solution for surface elevations. Two new subroutines were created to modularize the code associated with the gather and the scatter, respectively.

All remaining computations were performed in parallel by having each SPMD process restrict its computations to the nodes and elements in its local subdomain. This was effected by a simple change in loop bounds within any subroutines called from inside the timestep loop.

Since several of the computations require access to data from nearest neighbor nodes in the 2-D horizontal mesh, data for subdomain boundaries was explicitly passed as a 1-D array or 2-D array *via* PVM between processes of adjacent subdomains whenever changes to required variables occurred. The first dimension of the arrays that must be passed is determined by the number of nodes on the subdomain boundaries, and varies between SPMD processes. The second dimension is determined by whether or not the data at each node in the 2-D mesh extends into the vertical dimension; this dimension is constant for all SPMD processes, since the number of nodes in the vertical stretched grid is fixed. Two subroutines were added to modularize the necessary PVM code associated with passing a single 1-D array or a single 2-D array, respectively. A total of 26 calls to the 1-D message passer and 14 calls to the 2-D message passer were inserted into the timestep loop of the original code, in between subroutines that changed required variables. Although less frequent message passing could have been achieved by reorganizing some of the computations and by packing more than one array into each message, such reorganization was deliberately avoided in order to minimize programming effort.

In addition to the SPMD parallel processes, a distinct output process was utilized to perform all disk output, thus removing the output bottleneck which can hold up computation. The output process is identical to the SPMD processes, except that the body of the timestep loop has been removed and replaced by a call to a new subroutine that receives data to be output from each of the SPMD processes, then loads them into the data structures for the entire domain. The original output routine is subsequently called to perform the actual disk output. In place of the call to the original output subroutine, each SPMD process now calls a new subroutine that sends the data to be output from its local subdomain to the output process, and then continues with its computation of the next timestep.

Programming effort in the parallel port was thus very minimal. Eight new subroutines were created to package up all low-level calls to PVM message-passing and control routines: 1) spawn processes, 2) determine subdomains, 3) share a 1-D array of values with adjacent subdomains, 4) share a 2-D array of values with adjacent subdomains, 5) gather a vector from the other SPMD processes into the solver process, 6) scatter a vector from the solver back to the other SPMD processes, 7) Send output data to the output process, and 8) receive output data from the SPMD processes into the output process. Calls to these eight new subroutines were inserted into the mainline of the original code in between existing subroutine calls. The only significant change required to the existing code was the alteration of loop bounds within subroutines called from inside the timestep loop, in order to limit computations to the contiguous nodes and elements in the subdomain of each process. In addition, the call to output data was replaced by a call to send data to the output process.

4 Description of Timing Studies

Timings were performed on the PVM implementation as applied to a model of Lake Champlain. The domain was horizontally discretized into 627 nodes and 991 elements, and vertically discretized into a streched grid of 11 nodes (10 elements).

Timings were performed on the timestep loop run for 10 timesteps with output performed after timesteps 5 and 10. All timings were performed *via* calls to the C routine *gettimeofday*, which was linked to the Fortran-77 program. This wall-clock timer is necessitated in order to get fair and meaningful times of the PVM version, which *must* include message-passing delays which would be excluded in a CPU-clock timer.

In order to achieve meaningful speedups, timings were performed when both network and machine traffic were low, and were limited to two homogeneous PVM configurations: 1) nine 12 MHz SGI Iris Workstations on the same subnet, and 2) one 8-processor SGI 8D with four 32 MHz processors and four 40 MHz processors. No explicit control was exercised in placing separate processes onto separate processors on the multiprocessor.

Experiments were run with 1, 2, and 4 SPMD processes with and without a separate output process on both PVM configurations. Additionally, experiments with 8 SPMD processes with and without a separate output process (thus requiring a maximum of 9 processors, which exceeded the number available on the multiprocessor) were run on the workstation cluster. In all cases, subdomains were kept approximately uniformly sized. In cases where no separate output process was utilized, the solver process performed all disk output, so that the importance of the functional parallelism of the output process could be assessed.

Five replications of each timing experiment were performed. Low variability in the timings validates the assumption of low- load conditions (standard deviations of all experiments were less than 3 seconds for 10 timesteps).

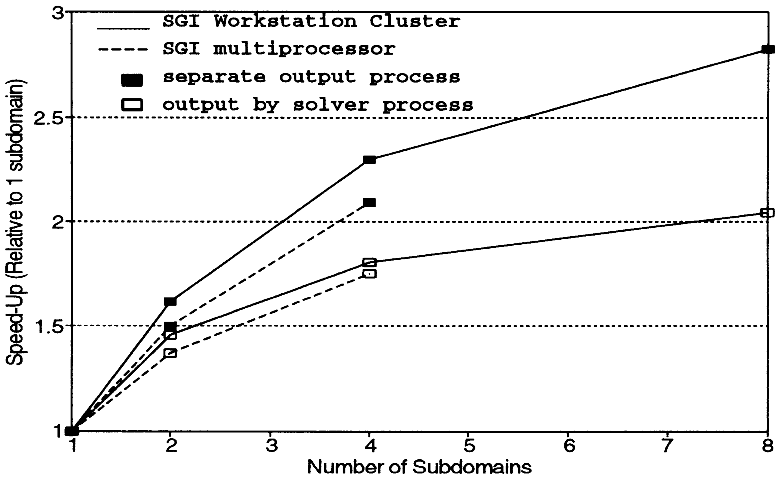

Figure 1: Min-based speedups for each PVM configuration with the Lake Champlain domain.

As discussed in [1], the minimum of each of the five replications was used in analysis of the results of the timings. The use of minimum timings taken during low-load conditions provides a consistent basis for comparison of parallel processes in a network-based distributed computing environment, and estimates an upper bound on performance.

5 Results and Discussion of Timing Studies

On both configurations, run time decreased monotonically as the number of subdomains (owned by separate SPMD processes running on separate processors) increased from 1 to 4 or 8. Interestingly, for both PVM configurations, the functional parallelization of disk output by the separate output process actually increased run times slightly when only one subdomain was used, but decreased run times increasingly as more subdomains were used. Presumably, the increase in run times with a separate output process and one subdomain is caused by the large overhead of passing the entire domain to the output process via PVM message-passing, outweighing the advantage of having the disk output done in parallel with subsequent computations. As the number of subdomains increases, the size of the subdomains (and hence the size of the output message each subdomain must send) decreases, and the functional parallelism of the output process becomes increasingly advantageous (figure 1).

Speedups were calculated relative to one subdomain, regardless of whether the output process was separated or not, and are shown in Figure 1. (Higher speedups were attained with the workstation cluster than with the multiprocessor because the communication to computation ratio is lower with slower processors.) A maximum speedup of 2.8 was achieved with 8 subdomains on the workstation cluster, while a maximum speedup of 2.1 was acheived with 4 subdomains on the multiprocessor. In both configurations, speedups were improved by the addition of the separate output process, especially for larger numbers of subdomains. For the 8 subdomains on the workstation cluster, the use of the separate output process improved speedup by 40%.

The speedups observed are less than linear and appear to have asymptotic behavior. Such asymptotic behavior is expected, since this PVM implementation has a serial bottleneck for the solution of the surface water elevations. The bottleneck limits the maximum attainable speedup and becomes increasingly dominant as the number of subdomains is increased. In order to remove this bottleneck, one could replace the solver by a domain decomposition algorithm, such as described in [4]. Previous studies [1] have shown that nearly linear speedup can be achieved with PVM through such an approach. However, an iterative solver is inherantly slower than a direct solver, and hence the attainment of better speedups may *not* be indicative of faster code.

Despite the low efficiency of adding processes, use of this PVM implementation can still reduce run times sufficiently to be useful in running long simulations that typically take many hours or days to complete. In a computing environment where there is a large amount of idle CPU time available on many machines, such as at the University of Vermont, efficiency is of little relevance. On the other hand, cutting down simulation time by a factor of 2-3 *is* significant and useful, enabling larger and more detailed simulations to be performed during low-load times, such as overnight, and may lower the probability that the simulation will be aborted due to an unexpected machine shut-down.

6 Summary and Conclusions

The PVM software system was used to successfully parallelize the majority of a complex 3-dimensional surface water flow model with *minimal* reprogramming (about 3 days effort). A combination of SPMD, functional parallelism and serial programming paradigms was employed. Despite a serial bottleneck and a high-frequency of message-passing, useful speedups were attained.

Although parallelism could undoubtedly be increased by redesigning and reprogramming parts of the original code, the programming effort that would be required would be significant. Many existing complex scientific codes remain serial because of the deterrant effect of the reprogramming effort often associated with parallelization. In this study, we have demonstrated that a simpler approach to parallel retrofits can yield useful speedups at virtually no cost, if expectations of computational efficiency of parallelization are relaxed.

7 Acknowledgements

M.J.E.'s participation in this project was made possible in part by a grant from the AAUW Educational Foundation.

References

[1] Eppstein, M.J. and Dougherty, D.E. "A Comparative Study of PVM Workstation Cluster Implementations of a Two-Phase Subsurface Flow Model", *Advances in Water Resources*, **17**:181-195, 1994.

[2] Laible, J.P. "On the Solution of the Three Dimensional Shallow Water Equations Using the Wave Equation Formulation",in *Computational Methods in Water Resources IX. Vol. 2: Mathematical Modeling in Water Resources*, (Ed. Russell, T. F., *et al.*), pp. 545-552, Proceedings of the Ninth International Conference on Computational Methods in Water Resources, Denver, 1992. Computational Mechanics Publications, Southampton and Elsevier Applied Science, London, 1992.

[3] Lamport, Leslie. "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", *IEEE Transactions on Computers*, **C-28**(9):690-691, 1979.

[4] Marini, L. D., and Quarteroni, A. "A Relaxation Procedure for Domain Decomposition Methods Using Finite Elements", *Numer. Math.*, **55**:575-598, 1989.

[5] PVM Version 3 source code and documentation may be obtained via email to **netlib@ornl.gov** with the subject message **send index from pvm3**.

[6] Sunderam, V. S. "PVM: A Framework for Parallel Distributed Computing", *Concurrency Practice and Experience*, **2**(4):315, 1990.