Computers & Geosciences 64 (2014) 96–103

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Parallelisation study of a three-dimensional environmental flow model

Fearghal O'Donncha, Emanuele Ragnoli*, Frank Suits

IBM Research, Ireland

ARTICLE INFO

Article history:
Received 1 October 2013
Received in revised form
29 November 2013
Accepted 5 December 2013
Available online 17 December 2013

Keywords:
MPI
Numerical modelling
Parallel computing
Ocean model

ABSTRACT

There are many simulation codes in the geosciences that are serial and cannot take advantage of the parallel computational resources commonly available today. One model important for our work in coastal ocean current modelling is EFDC, a Fortran 77 code configured for optimal deployment on vector computers. In order to take advantage of our cache-based, blade computing system we restructured EFDC from serial to parallel, thereby allowing us to run existing models more quickly, and to simulate larger and more detailed models that were previously impractical. Since the source code for EFDC is extensive and involves detailed computation, it is important to do such a port in a manner that limits changes to the files, while achieving the desired speedup. We describe a parallelisation strategy involving surgical changes to the source files to minimise error-prone alteration of the underlying computations, while allowing load-balanced domain decomposition for efficient execution on a commodity cluster. The use of conjugate gradient posed particular challenges due to implicit non-local communication posing a hindrance to standard domain partitioning schemes; a number of techniques are discussed to address this in a feasible, computationally efficient manner. The parallel implementation demonstrates good scalability in combination with a novel domain partitioning scheme that specifically handles mixed water/land regions commonly found in coastal simulations. The approach presented here represents a practical methodology to rejuvenate legacy code on a commodity blade cluster with reasonable effort; our solution has direct application to other similar codes in the geosciences.

1. Introduction

Numerical modelling has several advantages in the study of coastal ocean flow processes and events. Chief among these is the reduced cost and ease of deployment of a numerical model compared to field work or other methods of investigation. In addition, it is easier to configure a numerical model to investigate different flow conditions and scenarios. However, with the drive to model more realistic and detailed simulations, the computational demands of numerical solutions increase, due primarily to finer grid resolution and the simulation of a greater number of passive and active tracers. As a result, the practical ability of numerical models to solve real-world problems is constrained. Parallel computing allows faster execution and the ability to perform larger, more detailed simulations than is possible with serial code. The research reported here presents details on the porting of an existing coastal ocean model from serial code to parallel. This work is driven partly by a desire to model larger simulations in greater detail, but also to allow experimentation in more computationally demanding methods of data assimilation to improve the performance of real-time predictive modelling.

The model used for the study, Environmental Fluid Dynamics Code (EFDC), is a widely used, three-dimensional, finite difference, hydrodynamic model (Hamrick, 1992). The parallelisation adopts an efficient domain decomposition approach that theoretically permits deployment on a large cluster of machines; however the fundamental objective of our work centres on real-time simulation capabilities of a given model on a commodity blade system and not optimal scalability on an arbitrarily large system. We were therefore guided by the following requirements considered key to the success of the parallelisation effort and subsequent operation on similar cluster systems:

1. Limited changes to the large number of source files (approximately 50 000 lines of code), to avoid introducing computational errors.
2. Binary regression of the parallel model versus serial simulations, to ensure the simulation runs in parallel exactly as it ran serially. Even a small deviation could mask the presence of an error in the port.
3. Automation of the setup process for a parallel run to allow the originally setup serial models to run properly on the parallel code. This involves automatic generation of source code specific

* Corresponding author.
E-mail address: eragnoli@ie.ibm.com (E. Ragnoli).

0098-3004/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2013.12.006

to each parallel run of a model, to avoid manual effort and the introduction of errors.

Several parallel versions of numerical ocean models have already been described in the literature, and they have computational methods also used by other codes in the geosciences. Wang et al. (1997) present elements of the widely used Parallel Ocean Program (POP), while Beare and Stevens (1997) build on the parallelisation of the Modular Ocean Model (MOM). However, the fundamental structure of these models makes them more suitable for global, ocean-scale problems, and they are not as well-suited to the finer scale resolution of coastal water phenomena. A parallelisation study on the Princeton Ocean Model (POM) and the Regional Ocean Modelling System (ROMS) is discussed by Sannino et al. (2001) and Wang et al. (2005) respectively. A common feature of these models is the adoption of a split-explicit formulation of the equations governing vertically averaged transport. This representation permits easier parallelisation since global communication in the horizontal is eliminated. However the maximum computational timestep is constrained by the Courant–Friedrich–Levy restriction (Ezer et al., 2002), as opposed to the greater numerical flexibility provided by implicit approaches (Jin et al., 2000). De Marchis et al. (2012) presents details on a parallel code that adopts finite volume methods for the solution of the fundamental governing equations.

Among all branches of the geosciences, atmospheric modelling was one of the first to use parallel computers due to the intrinsic needs of both weather models that run in real-time, and climate models that operate in time scales of centuries. Coupling with ocean models similarly creates computational demands that benefit from parallel computation. Drake et al. (1993) present details on the parallel version of the NCAR Community Climate Model, CCM2. The parallelisation strategy decomposes the model domain into geographical patches with a message passing library conducting communication between segregated domains. Wolters and Cats (1993) describe the parallelisation strategy included in the HIRLAM model, a state-of-the-art system for weather forecasts up to 48 h, while Fournier et al. (2004) discuss aspects of deploying a spectral element atmospheric model in parallel. Michalakes et al. (1998) describe the parallelisation approach adopted for the widely used Weather Research and Forecast model.

In the following sections, the model is introduced along with a description of the computational schemes used to solve the governing equations. Section 3 discusses the parallelisation strategy adopted with particular emphasis on load balancing of the computation within an irregular coastal waterbody. Section 4 presents the parallel speedup and performance of the amended model; a case study analysis focuses on Galway Bay, on the West Coast of Ireland to enable a realistic assessment of practical gain. The conclusions and a discussion are found in section 5.

2. Model description

EFDC is a public domain, open source, modelling package for simulating three-dimensional flow, transport and biogeochemical processes in surface water systems. The model is specifically designed to simulate estuaries and subestuarine components (tributaries, marshes, wet and dry littoral margins), and has been applied to a wide range of environmental studies in the Chesapeake Bay region (Shen et al., 1999). It is presently being used by universities, research organisations, governmental agencies, and consulting firms (Ji, 2008).

The equations that form the basis for the EFDC hydrodynamic model are based on the continuity, Reynolds-averaged Navier–Stokes equations. Adopting the Boussinesq approximation for variable density fluid; which states that, density differences are sufficiently small to be neglected except where they appear in terms multiplied by g; the model governing equations can be expressed as

$$\frac{\partial \eta}{\partial t}+\frac{\partial Hu}{\partial x}+\frac{\partial Hv}{\partial y}+\frac{\partial w}{\partial z}=0 \tag{1}$$

$$\frac{\partial Hu}{\partial t}+\frac{\partial (Huu)}{\partial x}+\frac{\partial (Hvu)}{\partial y}+\frac{\partial (wu)}{\partial z}-fHv = -H\frac{\partial (g\eta+p)}{\partial x}-\left(\frac{\partial h}{\partial x}-z\frac{\partial H}{\partial x}\right)\frac{\partial p}{\partial z}+\frac{\partial\left(\frac{A_v}{H}\frac{\partial u}{\partial z}\right)}{\partial z} \tag{2}$$

$$\frac{\partial Hv}{\partial t}+\frac{\partial (Huv)}{\partial x}+\frac{\partial (Hvv)}{\partial y}+\frac{\partial (wv)}{\partial z}+fHu = -H\frac{\partial (g\eta+p)}{\partial y}-\left(\frac{\partial h}{\partial y}-z\frac{\partial H}{\partial y}\right)\frac{\partial p}{\partial z}+\frac{\partial\left(\frac{A_v}{H}\frac{\partial v}{\partial z}\right)}{\partial z} \tag{3}$$

where, η is water elevation above or below datum; u, v are the velocity components in the curvilinear orthogonal coordinates x and y with w representing the vertical component; H is total water depth ($=h+\eta$, where h=water depth below datum); f is Coriolis parameter; p is excess water column hydrostatic pressure; and ν_v is vertical turbulent viscosity.

The equations governing the dynamics of coastal circulation contain propagation of fast moving external gravity waves and slow moving internal gravity waves. It is desirable in terms of computational economy to separate out vertically integrated equations (external mode) from the vertical structure equations (internal mode) (Blumberg and Mellor, 1987). The external mode associated with barotropic, depth-independent, horizontal long wave motion is solved using a semi-implicit three time level scheme; the external mode computes surface elevation and depth-averaged velocities, $\overline{u}$ and $\overline{v}$. The internal mode, associated with baroclinic, fully three-dimensional, velocity components is solved using a fractional step scheme combining an implicit step for the vertical shear terms with an explicit discretisation for all other terms; the depth-averaged velocities computed in the external mode equations serve as boundary conditions to the computation of the layer integrated velocities. This approach solves the two dimensional depth-averaged momentum equations implicitly in time, hence allowing for the model's barotropic time step to equal the baroclinic time step. The primary limitation of this semi-implicit method is the introduction of an elliptic solver (preconditioned conjugate gradient) to solve implicitly for the free surface elevation solution. This has traditionally posed a problem in the efficient projection of model codes onto parallel computers due to the inherent non-local conditions of the solver (Griffies et al., 2000); this feature is addressed in further detail in the next section.

3. Parallelisation

EFDC is a Fortran 77 code originally designed for deployment on vector computers as opposed to distributed systems. The code was configured to achieve a degree of parallelisation on shared memory processors by directives inserted in the source specific to vectorised architectures. However, the existing vectorisation code is not of benefit for parallelisation on distributed memory systems. For performance comparable to vector systems, scalable cache based processors achieve speedup through massively parallel partitioning of the problem among many processors working concurrently (Griffies et al., 2001). The origins of our parallelisation study lie in the rapidly increasing computational requirements of environmental flow solutions, along with the general availability of distributed clusters, particularly blade systems. To enable real-time modelling of large

scale waterbodies and ensure maximum transferability across high performance computing (HPC) facilities, parallel deployment across these commonly available distributed clusters was desirable. Considering the practical nature of geoscientific modelling studies, the fundamental philosophy of the parallelisation effort focused on the question: If X number of computational nodes are available, what is the maximum simulation throughput that can be achieved from legacy serial code? In this case we are not seeking maximum scalability on a large number of nodes, nor are we tuning the number of nodes precisely to the problem at hand. Instead we address the common desire simply to run a given simulation much faster on a given cluster, and with minimal changes to the original serial code. All aspects of both the parallelisation itself and associated load balancing proceeded with this in mind.

Our partitioning scheme is motivated by the fact that ocean models tend to have a much greater horizontal than vertical length scale. We therefore adopt a two-dimensional domain decomposition approach in the horizontal plane; the vertical remaining unchanged. The work-flow is based on a data parallel execution model with all processors performing the same form of computation but on a different sub-domain. The only departure from this data parallel model is for the serial conjugate gradient scheme discussed later, when a single processor is designated to solve the implicit component of the model and distribute the solution across domains. Since conjugate gradient methods are often used in geophysical simulations, the results from our parallelisation study of this functional module may have direct application to other codes.

The domain decomposition strategy involved a number of additional considerations. Unlike some computational fluid dynamics problems involving a uniform volume of material, simulations of coastal ocean currents have a mix of land and water regions with very different computational costs. This complicates the task of balancing the work load to achieve maximum parallel performance, particularly when there is an irregular, serpentine coastline as in our model of Galway Bay. Fortunately the layout is fixed during our simulation, which means the partition need not be dynamically rebalanced, and a single, optimised partition can be used for the entire run. Although orthogonal recursive bisection is commonly used when the domain is fairly homogeneous (Williams, 1991), particularly with a large, power-of-two node count, the presence of large sections of land in our domain, plus the desire to use a relatively small and arbitrary number of processors on a commodity blade system, led us to choose a more general form of rectilinear partition that includes inactive tiles.

Normally the goal of load balancing is to make the work on each processor as equal as possible, but the presence of land in our simulation alters this by placing the focus on minimising the maximum work on any given processor. This means there may be processors with only a small amount of computation because they operate on cells that are mostly land, with only a small amount of water. But that is acceptable if, for that processor count, it results in the minimum time per timestep for the entire partition. Since the layout is rectilinear it will not be perfectly load balanced, but it will be much more balanced than a regular grid, and care is taken in the partitioning to benefit from inactive tiles that contain no water and therefore do not require an allocated compute processor. Fig. 1a shows a uniform rectilinear partition of the simulation into a 5×5 grid to run on 25 processors. Note that four of the tiles in the lower right are land and have no associated computational cost at all, and across the partition some tiles are entirely water while others are mostly land. Clearly this is a poorly load-balanced way to run the problem on 25 processors. In order to minimise added complexity to the code while achieving improved throughput, we retained the concept of a rectilinear grid, but allow both a range of tile sizes and the ability to mark some tiles as inactive. We call our algorithm for computing this optimal partition, LORP, for Locally Optimal Rectilinear Partition. The algorithm relies on estimates of the computational and communication costs in the simulation to find, for a given partition, the total number of active tiles, and the cost associated with the most expensive tile, using an empirically based estimate of the computation and communication times derived from a number of runs with different partitions. A key component of the LORP algorithm is an estimate of the time per timestep for a given tile based on the following expression:

$$TimePerTimestep_i = a*WaterArea_i + b*BoundaryLength_i + c*TileArea_i + d \quad (4)$$

This is an expression for the approximate time a given tile will take per timestep during parallel execution, determined by the computational cost associated with its water area, combined with the communication cost of its boundary length and total area including land, plus some fixed overhead. The coefficients for a given model are tuned based on empirical results from preliminary runs of different partitions, where timers inserted in the code allow measurement of the computation and communication costs associated with different domain sizes. Note that when a tile is inactive it is not executed at all and the overhead term does not apply. The model for the timing need not be exact, but it needs to be accurate enough to allow a reasonable estimate of the time per timestep for the slowest tile in a given partition. A primary aspect of a good partition is one for which there are many tiles containing no water, which are certain to have no cost associated and therefore do not need a compute node assigned to them.

For a given number of desired active partitions, A, the LORP algorithm explores a range of possible partitions with tile count much larger than A, to allow for possible inactive tiles that contain no water, and hence no computation. The only requirement is that the final partition must contain exactly A active partitions. Thus a

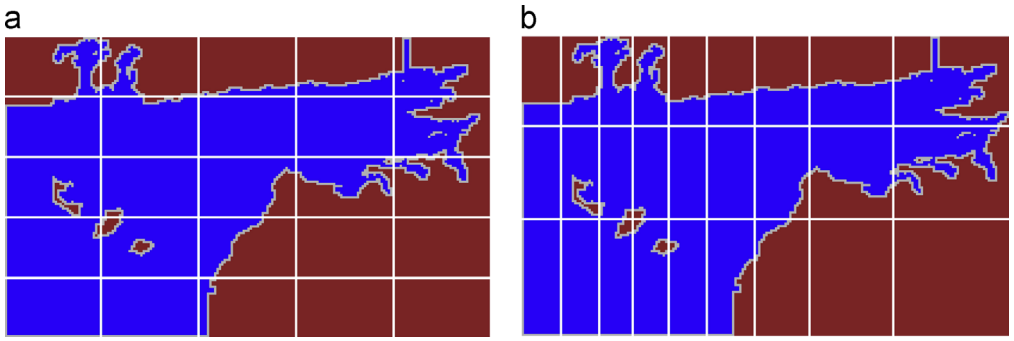

Fig. 1. Rectilinear domain decomposition demonstrating (a) a fixed partitioning assigning equal number of cells to each processor (LHS) and (b) locally Optimal Rectilinear Partitioning (RHS). Active water cells are shown in blue while red denotes inactive land cells. Note that (a) has four inactive tiles consisting entirely of land, while (b) has two, but they are larger. The entire domains consist of 196×130 grid cells encompassing an area of 59×39 km. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

wide range of trial partitions is considered of the form $M \times N$, where $A \leq M \times N \leq 2A$ to allow for some inactive tiles in the partition. For each trial $M \times N$ partition, a greedy algorithm is applied to divide the total cost in each vertical and horizontal band equally based only on water content. Instead of exploring the full range of options for subdividing the original grid into $M \times N$ partitions, which would be prohibitive, LORP does a local exploration of options around that partition by successive adjustments of each "wall" in the partition in smaller powers of two from an initial step size. If any adjustment of a wall finds an improved partition, with lower maximum time per timestep at the worst tile, that partition is chosen and the search continues in a greedy manner. If no adjustments find improvement at the given step size, the step size is halved and the search repeated until the step size reduces to zero and the locally optimal rectilinear partition, LORP, is found.

Since this algorithm accepts any downward step in cost, it is greedy and is not expected to find the true globally optimal partition, but it does explore the optimisation landscape around the starting partition to find one that is locally optimal. LORP is therefore in the class of load balancing algorithms based on choosing an initial nearly balanced configuration, then seeking a more efficient partition similar to it. There have been several variations on this approach since the early work of Fiduccia and Mattheyses (1982) and later in combination with simulated annealing by Martin and Otto (1996), with the main difference that the cuts in the domain that define the partition are allowed to adjust in a local search for improvement. Since LORP requires a computationally demanding search over partition candidates, it is written in C++ rather than a high level scripting language.

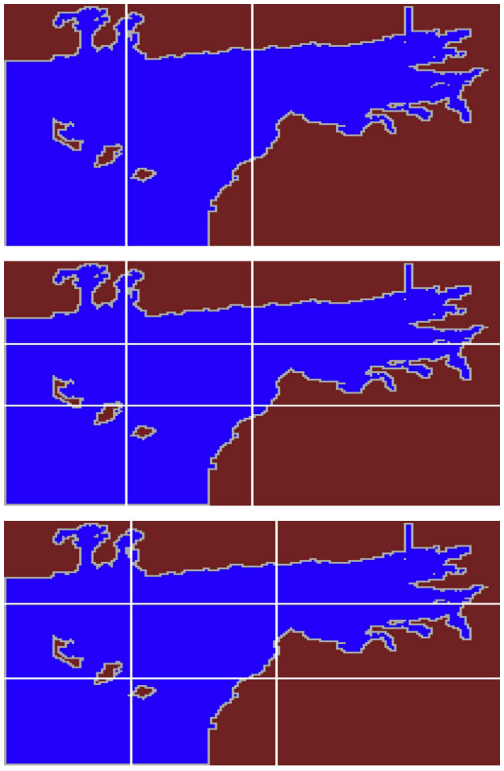

Fig. 2. Schematic of the LORP algorithm seeking a partition with 8 active nodes based on an initial 3×3 rectilinear partition. LORP initially conducts two vertical cuts to divide the domain laterally into sections with equal water (top); then cuts the domain vertically into three horizontal slabs with equal water (middle). Since the result is imbalanced and has nine tiles containing water rather than the desired eight, LORP refines the partition boundary and discovers an acceptable solution with eight active nodes and better load balancing, shown in bottom figure.

For the simulations in this paper the algorithm takes approximately 30 s on a typical laptop, which is negligible overhead compared to the reduced time of the simulation from culling land-only sections and better load balancing. The cost will increase with a large partition, but again this paper targets the node counts of the smaller cluster systems. The step size therefore can be chosen for the particular problem based on the time allocated to find an improved partition and the potential improvement in throughput for the simulation run.

As an example, Fig. 2 shows the steps in the LORP algorithm to find a well balanced scheme to run on eight nodes. The model is Galway Bay and like many coastal regions it includes a complex mixture of water and land. Since the target is eight active tiles, all combinations of grid layouts with 8–16 tiles are considered, e.g. 1×8, 2×4, 2×5, 3×4, etc. The goal is to end up with exactly eight tiles containing roughly the same amount of water cells. Fig. 2 shows the exploration of the 3×3 option. First two vertical cuts are made to divide the domain into three regions of equal water (Fig. 2a). This is a well defined partitioning method, but the end result may not be the best choice – hence it is "greedy." Next the domain is divided into three horizontal bands also with equal amounts of water, as shown in Fig. 2b. The resulting 3×3 partition does not, in fact, have equal amounts of water in each tile, and the small amount of water in the lower right tile requires nine rather than eight nodes for execution. Given this initial guess at a balanced partition, LORP then systematically adjusts each of the partition walls in progressively smaller steps to seek a better balanced partition with exactly eight active tiles. For a 3×3 tiling, LORP finds an optimal partition with eight active tiles shown in Fig. 2c. There are now exactly eight tiles containing water, and the amount of water in each is well balanced. It is not perfect because perfect load balancing cannot be achieved in general with rectilinear partitioning, but it is greatly improved over the initial guess, and it contains the desired number of active regions. Once the best partition is found for the 3×3 case, all other feasible options such as 2×5 are also explored, and the overall best solution is selected for the run. For the runs reported here, and for a target of 25 processors, the LORP partition is 9×3 with two inactive tiles, as shown in Fig. 1b.

In the parallel implementation, each partition is extended by two ghost cells in all directions possessing information on the neighbouring partition; this enables computations to proceed independently with minimum communication and synchronisation. At the end of each computation step the variables stored in these ghost cells must be updated by receiving values from neighbouring partitions. A further advantage of this domain decomposition approach is that the memory requirements of the code are drastically reduced since each processor needs information only on a sub-domain of the model. In addition, this strategy guarantees complete binary regression of the code relative to serial.

Fig. 3 describes the procedure adopted to incorporate the domain decomposition and parallel code pre-processing into the existing serial code. The LORP decomposition operates on a serial model grid, with the only required input being the number of available processors. After calculating an initial best guess and refining that guess to determine the most optimal nearby partition, LORP outputs information on the size of each partition to the Python pre-processing module. The additional module processes the existing input files and domain geometry, generating all required input files and memory allocation functions for the child partitions. Since this phase involves code generation and parsing of text, it is convenient to do it in a scripting language such as Python. Model compilation and execution follows, then the final stage of the simulation involves the reconciliation of model output files from each processor into a single array of output files. The entire pre-processing, compilation, execution and post-processing is done by a simple bash script to avoid manual

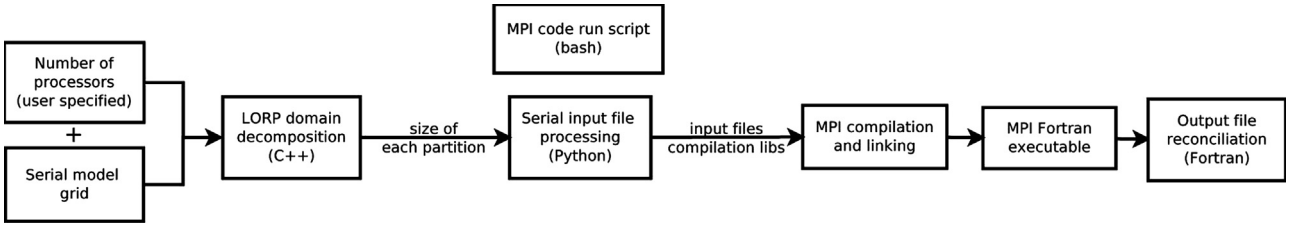

Fig. 3. Flowchart of the entire MPI code pre-processing, simulation and post-processing. The only required user input is number of available processors; optimal partition is computed by LORP, the output of which is used by a Python pre-processing module to amend all input files and memory allocation function as appropriate. After simulation the final stage is reconciliation of output files from each processor into equivalent output from serial code. The entire routine is conducted by means of a bash script to minimise complexity.

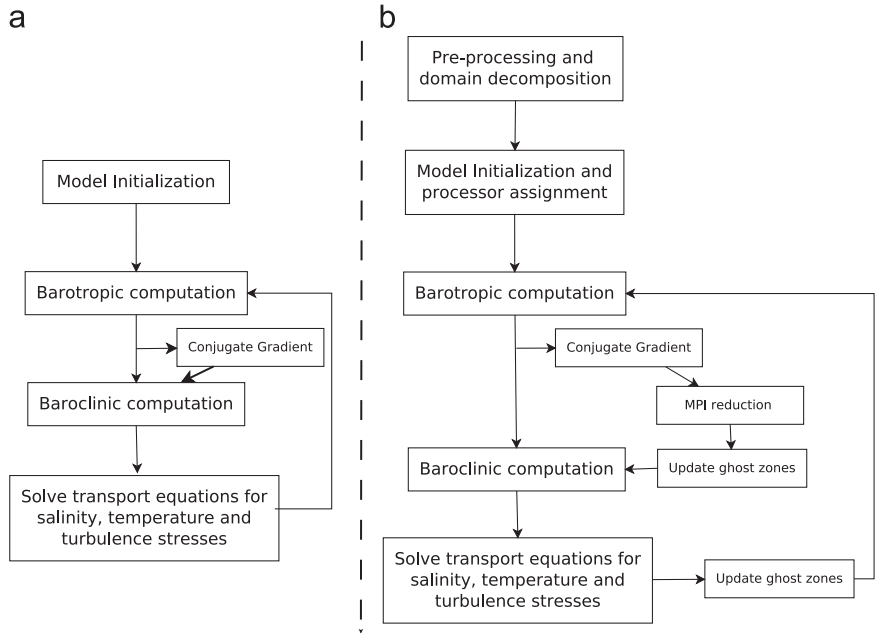

Fig. 4. Schematic outlining model progression over a single time step for (a) serial code and (b) MPI parallel code.

errors and to enable future deployment of the parallel code by users with no parallel computing expertise.

Once the partition has been found and the corresponding code generated and compiled, we can then discuss the work performed and communication required during a parallel simulation. Communication between processes is by means of a standard message passing library, MPI (Snir et al., 1995), to ensure portability of the code; in particular, with regard to the target deployment on a cluster of commodity machines. The communication protocol can be divided into two categories: point-to-point communication between ghost zones and global communication to compute the implicit free surface elevation solver. The computational sections of the code then sequentially resolve the external mode equations, the internal mode equations, and the transport equations for salinity, temperature, turbulence intensity and turbulence length scale. The final stage of the computational step involves a point-to-point communication to update ghost zone arrays prior to the next timestep. Fig. 4 presents a schematic outline of both the serial and parallel version of the program.

The solution of the external mode equation required particular attention due to the presence of an elliptic equation for the free surface displacement. Computationally, the system of equations is solved by a reduced system conjugate gradient scheme; iterations continue until the sum of the squared residuals is less than a specified tolerance. This involves a summation across the entire domain; hence, the data dependencies at each iteration are global and cannot be treated independently (Hageman and Young, 2012). Two approaches were considered to address the global communication of the solver:

- the parallel solution of the conjugate gradient on each processor requires a global MPI reduction call and a point-to-point communication at each iteration of the solver. Typically 10–40 iterations are required for convergence (Hamrick, 1996).
- an alternative approach that avoids the need for communication at each iteration is to have all processors communicating the required variables to a single processor. The solver then proceeds in an identical manner to the serial code after which the computed value of surface elevation is distributed across all partitions.

A drawback of the second approach is that gathering all variables on a single processor introduces a large memory footprint as an array equal to the size of the entire domain must be allocated for each relevant variable. As the model domain becomes large and the number of processors increases into the hundreds, this approach can be prohibitive in memory terms. To reduce the impact of the demands on memory, a hybrid approach incorporating Fortran 90 features into the legacy Fortran 77 code was adopted. Since the allocation of sufficient memory to store the gathered data on the processor charged with solving the conjugate gradient algorithm was unavoidable, the objective was to localise the memory allocation to this single processor. This required the usage of dynamic memory allocation not available in Fortran 77; hence, the subroutine

associated with the serial solution of conjugate gradient was rewritten in Fortran 90 and associated memory allocated to a single processor only.

4. Performance

All performance tests were conducted on a local commodity blade cluster of five nodes. Each compute node had a X5690 hex-core processor, with clock speed of 3.47 GHz and 12 MB of cache; the nodes are connected by a 1 GBit/s Ethernet network. Parallel simulations were configured to deploy on the smallest number of blades possible to minimise unnecessary network communication.

Experiments have been performed on a typical coastal region application, Galway Bay, to investigate the performance of realistic models on the cluster. Previous studies have applied and validated fundamental model predictions against measured oceanographic data for this bay (O'Donncha et al., 2012a, 2012b). The model domain consisted of 196×130 grid cells in the horizontal at a resolution of 300 m, along with 10 layers in the vertical. Benchmarking of model performance focused on strong scaling (i.e. the overall spatial domain size remains constant regardless how many processors are used to solve it), since it is the most relevant measure if turnaround time for a fixed problem is important (Deane et al., 2006). In practical hydro-environmental studies, the size of the problem is established based on physical considerations, and the desire is to solve in the minimum time possible using as many processors as available on a given cluster. Weak scaling is of interest because it corresponds to large problems running on more processors, where scalability is easier to maintain than in the strong scaling case of a given problem running on more processors. Fig. 5a presents the number of timesteps computed per minute simulation time for three configurations of the MPI code: (1) a fixed domain decomposition with the model domain divided into equal number of partitions in the x and y direction; (2) domain decomposition performed by LORP with the non-local conjugate gradient routine performed in serial and (3) LORP decomposition with conjugate gradient routine proceeding in parallel with MPI communication between processors at each iteration.

Accuracy of each configuration was checked after each simulation. This involved a binary, single precision, write-to-file of eight fundamental variables (water level, three components of velocity, turbulent kinetic energy and length scale, salinity and temperature) for the first 5000 timesteps. Results were compared to serial with any deviations considered unacceptable.

Load balancing required specific attention depending on the particular configuration adopted. As a qualitative check on performance and bottlenecks all model execution times were analysed using the visual performance tool, jumpshot (Zaki et al., 1999). This served to identify obviously inefficient decompositions and also fine tune coefficients used in the LORP algorithm. In the case of configuration (2), in which the master processor is tasked with solving the conjugate gradient routine in serial each timestep, load balancing is more complex. In this case it can be beneficial to further balance the load on the different processors so as that the master processor has a lower load as demonstrated by Wu et al. (2013). However, in our situation running conjugate gradient in parallel gave much better performance which rendered further finetuning of configuration (2) academic. If this was not the case then LORP would have to be modified to account for this imbalance.

Results demonstrate a number of interesting features, namely the superior performance of the LORP decomposition relative to the more elementary equal partitioning often adopted in regional ocean models (Wang et al., 2005), and the superior performance of a parallel deployment of the conjugate gradient routine despite the increased number of MPI communication calls involved.

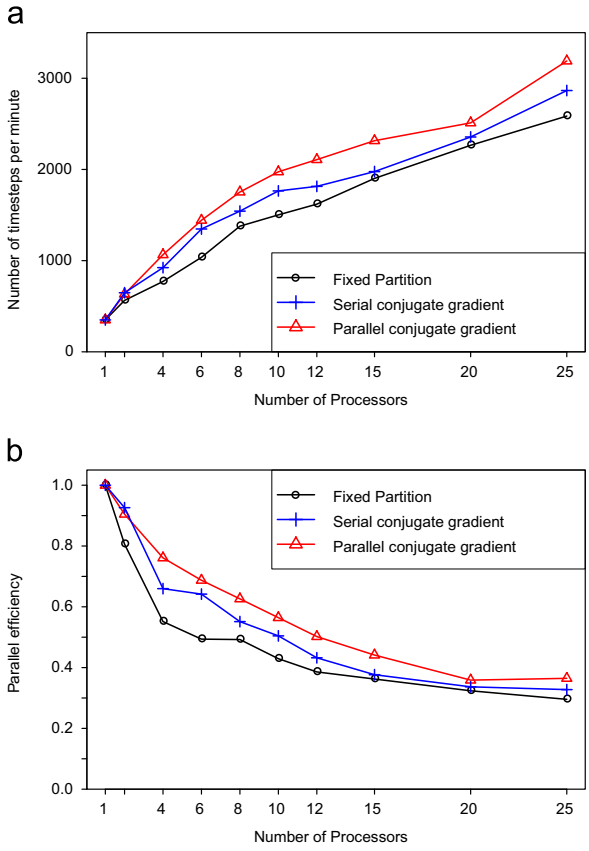

Fig. 5. (a) Number of timesteps per minute performed by MPI code varying the number of processors and (b) efficiency of the parallel MPI EFDC code. Three configurations of the MPI code presented: domain split into number of equal partitions; domain decomposition performed by LORP and conjugate gradient routine computed in serial; and LORP decomposition with conjugate gradient scheme computed in parallel.

Adopting the best performing configuration provides 3190 timesteps per minute running on 25 processors relative to 350 timesteps when the model is deployed in serial. Considering a model timestep of 4 s, this corresponds to one day of simulation time in 6.77 min wallclock time; a comparable simulation in serial requires 61.7 min. Considering current demands on real-time modelling capabilities providing accurate, contiguous forecasting in a 48–72 h window the benefits of this speed-up are clear.

To quantitatively compare the different parallel configurations, the parallel efficiency (E), of the code was computed:

$$E = \frac{1}{N_p}\left[\frac{T_1}{T_p}\right] \tag{5}$$

where T_1 is the time of simulation for serial code; T_p simulation time on p processors; and N_p represents number of processors

Fig. 5b presents the parallel efficiency of the code. Considering the implicit solver that is necessary during code execution and the constraints imposed by strong scaling, the results of scaling tests demonstrate greatly increased throughput from the model. If weak scaling were considered, superior results would be expected as the computational cost would stay fixed with increasing number of processors. Alternatively, choosing a larger initial domain would provide better scaling results as computation would account for a greater portion of the total simulation time. However, the objective of the research was primarily to maximise throughput for a typical fixed-size model deployment, which the parallel strategy successfully accomplishes, as demonstrated by Fig. 5a.

5. Discussion and conclusions

This study presents details on the parallelisation of a widely used environmental flow model. Preliminary results demonstrate that considerable speedup can be achieved on a distributed cluster by adopting a pragmatic approach to the parallelisation effort, with a load-balanced domain decomposition based on the underlying numerical algorithms. Note that this study presents details only on the hydrodynamic simulation itself, and not more computationally demanding aspects of a simulation. Introducing a greater computational workload, particularly for real-time forecasts, when simulating water quality constituents and density driven circulation, will derive further benefits from parallelisation.

A key component of the pipeline is the domain decomposition module that computes an optimally balanced division of load with no user inputs required other than the number of processors available. The LORP module computes the computation and communication load of each partition and eliminates all 'dead' partitions containing only land from computations. Analysis of parallel model speedup demonstrates that this pre-processing routine significantly improves model performance. Deploying both a fixed, equal partitioning and an equivalent optimal decomposition on 25 processors yields an increase in speedup of 23% for the latter model configuration. Current state-of-the-art parallel ocean models decompose the model domain into a prescribed number of equal sized domains (Wang et al., 2005). The limitations of this in a coastal simulation are the introduction of computational inefficiencies due to poor load balancing and the possibility of processors being assigned pure land involving no computation. Some developments in this area exist, with the NEMO ocean model providing capabilities to remove land-only domains from computations. However, there exists no automated feature to remove these land partitions, which means the user needs to select the decomposition and then independently determine how many partitions contain only land (Reid, 2010). The domain decomposition approach discussed here represents an improvement combining both improved computational performance and minimal user interaction. This is particularly beneficial in coastal regions where irregular coastlines introduces a significantly more complex load balancing challenge than in the open ocean.

In all branches of the geosciences, it is desirable to separate the computational science layer from the natural science layer. This research presents a complete system to deploy a serial code model on a distributed cluster system. No additional knowledge is required of the user to run the code in parallel with all input files untouched. A comprehensive methodology is presented that: minimises existing code changes; takes advantage of modern Fortran features in legacy code; relies on convenient Python scripting for text processing and code generation, but C++ for the performance-demanding LORP partitioning. The entire system is enveloped in a bash script to minimise user interaction and enable convenient transportation to alternative computing platforms.

The solution of implicit algorithms represents an inherent challenge to parallel code design (Griffies et al., 2000). This study considers two approaches to the problem: adopting a serial computation methodology that reduces communication to a single gather/scatter routine, and a parallel version that involves a global sum and point-to-point communication at each iteration of the solver. Simulations illustrate that despite the expected increases in communication latency, the parallel solver performs better for all processor configurations. This also removes a serious obstacle to realised scale-up in that the serial solver would represent a growing constraint on performance as the scale of the problem or the number of processors is increased. This is evidenced by Fig. 5(b) which shows the parallel efficiency of the serial conjugate gradient model code continues to decrease as the number of processors increases to 25 while that of the parallel conjugate gradient tends to asymptote These results demonstrate the viability of deploying an ocean model with an implicit free surface solver in parallel. Previous studies have raised concern over the difficulties involved due to the inherent non-local communications (Griffies et al., 2000, 2001). Considering the acceptable scalability presented in this paper, combining the advantages of the superior numerical performance of implicit algorithms with parallel deployment is a competitive alternative to the more widely used explicit methods adopted in ocean modelling.

Further improvements in scalable model performance are possible from structural reorganisation and optimisation of the serial model code (Beare and Stevens, 1997). EFDC was originally developed for deployment on vector supercomputers with strong emphasis on shared memory; hence, the performance of the code on modern parallel systems with fast CPUs and distributed memory will not fully exploit resources. Adopting optimised memory access, loop structuring and libraries has been demonstrated to significantly improve simulation time of existing codes deployed on modern systems (Wang et al., 1997). However, in practice, recoding for optimal performance could lead to changes so extensive that the task becomes comparable to a complete rewrite of the code representing a large time investment. The approach presented here represents a pragmatic way to rejuvenate legacy code and enable high throughput computation on a commodity blade cluster. Considering that the majority of existing ocean modelling codes are Fortran – with legacy Fortran 77 codes comprising the bulk of ocean models (Griffies et al., 2000) – this research has wide application.

References

Beare, M., Stevens, D., 1997. Optimisation of a parallel ocean general circulation model. Annales Geophysicae, vol. 15. Springer, pp. 1369–1377.

Blumberg, A.F., Mellor, G.L., 1987. A description of a three-dimensional coastal ocean circulation model. Coast. Estuar. Sci. 4, 1–16.

De Marchis, M., Ciraolo, G., Nasello, C., Napoli, E., 2012. Wind-and tide-induced currents in the stagnone lagoon (sicily). Env. Fluid Mech. 12 (1), 81–100.

Deane, A., Brenner, G., Emerson, D.R., McDonough, J., Tromeur-Dervout, D., Satofuka, N., Ecer, A., Periaux, J., 2006. Parallel Computational Fluid Dynamics 2005: Theory and Applications. Elsevier.

Drake, J., Flanery, R., Walker, D., Worley, P., Foster, I., Michalakes, J., Stevens, R., Hack, J., Williamson, D., 1993. The message passing version of the parallel community climate model. In: Proceedings of the Fifth ECMWF Workshop on Use of Parallel Processors in Meteorology, pp. 500–513.

Ezer, T., Arango, H., Shchepetkin, A.F., 2002. Developments in terrain-following ocean models: intercomparisons of numerical aspects. Ocean Model 4, 249–267.

Fiduccia, C.M., Mattheyses, R.M., 1982. A linear-time heuristic for improving network partitions. In: 19th Conference on IEEE Design Automation, pp. 175–181.

Fournier, A., Taylor, M.A., Tribbia, J.J., 2004. The spectral element atmosphere model (SEAM): high-resolution parallel computation and localized resolution of regional dynamics. Mon. Weather Rev. 132 (3), 726–748.

Griffies, S., Boning, C., Bryan, F., Chassignet, E., Gerdes, R., Hasumi, H., Hirst, A., Treguier, A., Webb, D., 2000. Developments in ocean climate modelling. Ocean Model 2 (3), 123–192.

Griffies, S., Pacanowski, R., Schmidt, M., Balaji, V., 2001. Tracer conservation with an explicit free surface method for *z*-coordinate ocean models. Mon. Weather Rev. 129 (5), 1081–1098.

Hageman, L.A., Young, D.M., 2012. Applied Iterative Methods. Dover Publications. com.

Hamrick, J., 1992. A Three-Dimensional Environmental Fluid Dynamics Computer Code: Theoretical and Computational Aspects. Technical Report. Virginia Institute of Marine Science.

Hamrick, J.M., 1996. User's Manual for the Environmental Fluid Dynamics Computer Code. Technical Report. Virginia Institute of Marine Science.

Ji, Z., 2008. Hydrodynamics and Water Quality: Modeling Rivers, Lakes, and Estuaries. John Wiley & Sons.

Jin, K., Hamrick, J.M., Tisdale, T., 2000. Application of three-dimensional hydrodynamic model for Lake Okeechobee. J. Hydraul. Eng. 126 (10), 758–771.

Martin, O.C., Otto, S.W., 1996. Combining simulated annealing with local search heuristics. Ann. Oper. Res. 63 (1), 57–75.

Michalakes, J., Dudhia, J., Gill, D., Klemp, J., Skamarock, W., 1998. Design of a next-generation regional weather research and forecast model. Towar. Teracomput., 117–124.

O'Donncha, F., Ragnoli, E., Katrinis, K., Hartnett, M., 2012a. Characterising observed radial patterns within a bay using HF radar and numerical model simulations. In: 10th International Conference on Hydroinformatics.

O'Donncha, F., Ragnoli, E., Zhuk, S., Suits, F., Hartnett, M., 2012b. Surface flow dynamics within an exposed wind-driven bay: combined HF radar and model simulations. In: IEEE/MTS Oceans 2012.

Reid, F., 2010. The NEMO ocean modelling code: a case study. In: Cray User Group meeting.

Sannino, G., Artale, V., Lanucara, P., 2001. An hybrid OpenMP-MPI parallelization of the Princeton ocean model. In: Parallel Computing, Advances and Current Issues. Proceedings of the International Conference, ParCo2001. Naples, Italy. Imperial College Press, London.

Shen, J., Boon, J.D., Kuo, A.Y., 1999. A modeling study of a tidal intrusion front and its impact on larval dispersion in the James River Estuary, Virginia. Estuaries 22 (3), 681–692.

Snir, M., Otto, S.W., Walker, D.W., Dongarra, J., Huss-Lederman, S., 1995. MPI: The Complete Reference. MIT Press.

Wang, P., Katz, D.S., Yi, C., 1997. Optimization of a parallel ocean general circulation model. In: Proceedings of Supercomputing 97. San Jose, CA.

Wang, P., Song, Y.T., Chao, Y., Zhang, H., 2005. Parallel computation of the regional ocean modeling system. Int. J. High Perform. Comput. Appl. 19 (4), 375–385.

Williams, R.D., 1991. Performance of dynamic load balancing algorithms for unstructured mesh calculations. Concurr.: Pract. Exp. 3 (5), 457–481.

Wolters, L., Cats, G., 1993. A parallel implementation of the Hirlam model. In: Parallel Supercomputing in Atmospheric Science, Proceedings of the Fifth ECMWF Workshop on the Use of Parallel Processors in Meteorology. World Scientific Publ. Citeseer. pp. 486–499.

Wu, Y., Li, T., Sun, L., Chen, J., 2013. Parallelization of a hydrological model using the message passing interface. Env. Model. Softw. 43, 124–132.

Zaki, O., Lusk, E., Gropp, W., Swider, D., 1999. Toward scalable performance visualization with jumpshot. Int. J. High Perform. Comput. Appl. 13 (3), 277–288.