

Optimization of the POLCOMS Hydrodynamic Code for Terascale High-Performance Computers

Mike Ashworth
CCLRC Daresbury Laboratory, Warrington, WA4 4AD, UK
Email: m.ashworth@dl.ac.uk

Jason T. Holt and Roger Proctor
Proudman Oceanographic Laboratory, Bidston Observatory,
Birkenhead, CH43 7RA, UK

Abstract

Advances in computational science are closely tied to developments in high-performance computing. We consider the case of shelf sea modelling where models have been growing in complexity and where model domains have been growing and grid resolutions shrinking in pace with the increasing storage capacity and computing power of high-end systems. Terascale systems are now readily available with performance levels measurable in TeraFlop/s and memories counted in TeraBytes. The scientific case is now being made for regional models at 1km resolution, allowing the accurate representation of eddies, fronts and other regions containing steep gradients. The hydrodynamic model is increasingly being coupled with other models in multidisciplinary studies e.g. ecosystem modelling and wave modelling. We show that the performance attainable from the POLCOMS hydrodynamic code is measurable at about 0.5 TeraFlop/s on an IBM p690 cluster with 1024 processors. The scalability on this system and others is excellent up to 1000 processors. We describe a wide range of optimisations which have together enabled this code to reach these performance levels.

This is a Technical Report from the HPCx Consortium.

Report available from <http://www.hpcx.ac.uk/research/publications/HPCxTR0415.pdf>

© UoE HPCx Ltd 2004

Neither UoE HPCx Ltd nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

<i>1</i>	<i>Introduction</i>	<i>3</i>
<i>2</i>	<i>Memory optimization for single processor performance</i>	<i>3</i>
<i>3</i>	<i>Grid Partitioning</i>	<i>4</i>
<i>4</i>	<i>Boundary Exchange</i>	<i>8</i>
<i>5</i>	<i>Terascaling the POLCOMS code</i>	<i>11</i>
<i>6</i>	<i>Summary</i>	<i>12</i>
	<i>References</i>	<i>13</i>
	<i>Publication history</i>	<i>13</i>

1 Introduction

The Proudman Oceanographic Laboratory Coastal Ocean Modelling System (POLCOMS) has been developed to tackle multi-disciplinary studies in coastal/shelf environments [4]. The central core is a sophisticated 3-dimensional hydrodynamic model that provides realistic physical forcing to interact with, and transport, environmental parameters. Integrating from ocean to coast, or vice versa, biological production and the fate of contaminants can be determined.

The hydrodynamic model is a 4-dimensional finite difference model based on a latitude-longitude Arakawa B-grid in the horizontal and S-coordinates in the vertical. Conservative monotonic PPM advection routines are used to ensure strong frontal gradients. Vertical mixing is through turbulence closure (Mellor-Yamada level 2.5).

In order to study the coastal marine ecosystem, the POLCOMS model has been coupled with the European Seas Regional Ecosystem Model (ERSEM) [5]. Studies have been carried out, with and without the ecosystem sub-model, using a shelf-wide grid at 12km resolution (a grid size of approx. 200 x 200 x 34). In order to improve the simulation of marine processes, we need accurate representation of eddies, fronts and other regions of steep gradients. The next generation of models will need to cover the shelf region at approximately 1km resolution.

2 Memory optimization for single processor performance

Until the appearance of microprocessor based systems as serious competitors in the HPC market, vector processors, as exemplified by (but not limited to) the Cray supercomputers, dominated the scene. Many scientists became skilled in adapting and writing codes for efficient execution on vector processors and many important codes in use today still carry the legacy of this. The code is typically organised with long, unit-stride inner loops to allow the compiler to generate long vectors. Such is the benefit of long vectors, that conditional execution is often implemented by computing everywhere and masking out unnecessary data with a vector mask. Intermediate results are often promoted from scalars to vectors increasing the amount of memory required. The priority is to generate unit-stride, vectorisable loops.

Different techniques are required for programming the cache-based microprocessors that are to be found in most of today's high performance PCs, workstations and massively parallel processors (MPP). Processor speeds have out-paced memory access speeds to such an extent that the major factor limiting performance is now the memory access pattern. Unit-stride access is still important but for a different reason; that is, to ensure optimum use of cache, rather than to optimise processor performance. Inner loops should be small, so that their data fits within cache; intermediate results should be scalars so that they remain in cache or in registers; and vector masking should be discarded in favour of computing only where necessary. Unrolling inner loops often helps performance, whereas before it only inhibited vectorisation.

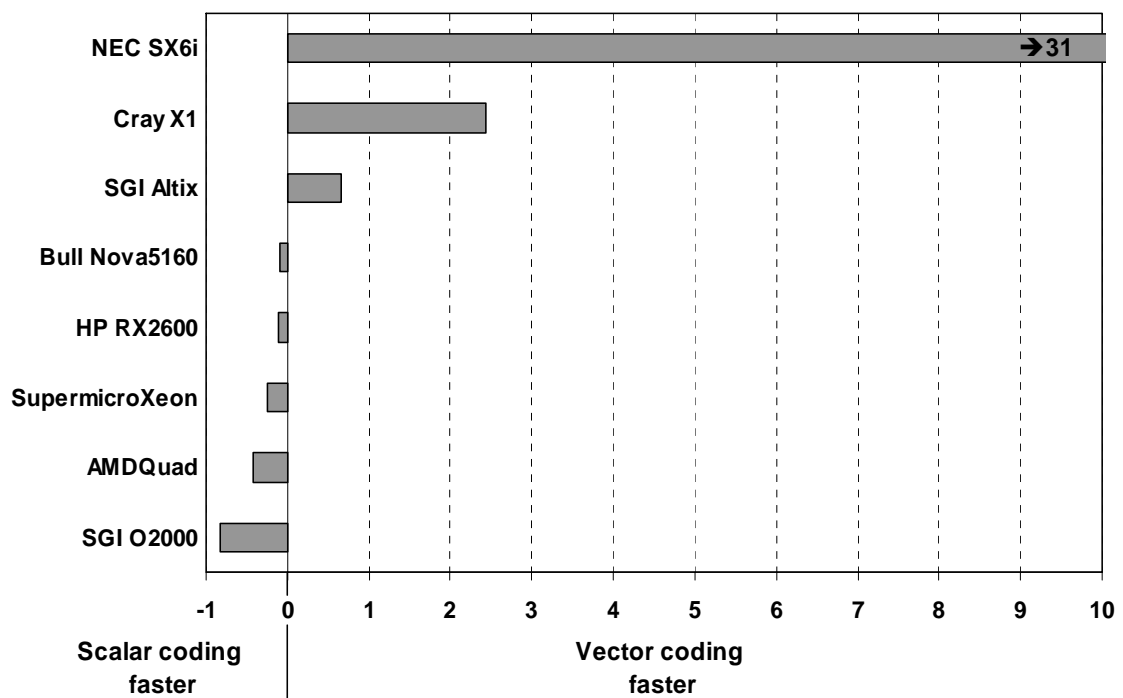


Figure 1: (Speed-up minus 1) of the total execution time of the Shelf Edge Model for the scalar and vector codes on a range of high performance systems. Zero corresponds to equal performance.

The original vectorised code had data arrays organized with the horizontal dimensions first (contiguous in memory) like $a(i,j,k)$ (where i and j are the longitude and latitude indices and k is the vertical index) or even with the horizontal dimensions collapsed into a single array index: $a(ij,k)$. We have found [after 1] that on cache based systems there is considerable advantage in restructuring the data arrays with the smallest index, the vertical dimension, first: $a(k,i,j)$. Results from a kernel of the Shelf Edge Model (a close relative of POLCOMS) written in both $a(ij,k)$ form (vector) and in $a(k,i,j)$ form (scalar) are shown in figure 1. A performance ratio in favour of the vector coding of between 2.5 and 32 is found for vector processors whereas the advantage of using the scalar coding on microprocessor systems is up to a factor of two.

3 Grid Partitioning

Ocean and shallow sea models operate on a three-dimensional physical domain but one that is very much thin and flat. The vertical dimension typically extends to at most a few km whereas the horizontal dimensions can be several thousand km. When considering how to distribute the data across a distributed memory computer system, it is therefore logical to partition on two dimensions only. This has the further advantage that many physical processes (e.g. convection) take place within a water column, and it is advantageous for these parts of the code if the data for a whole water column is situated in the memory (and especially the cache) of a single processor.

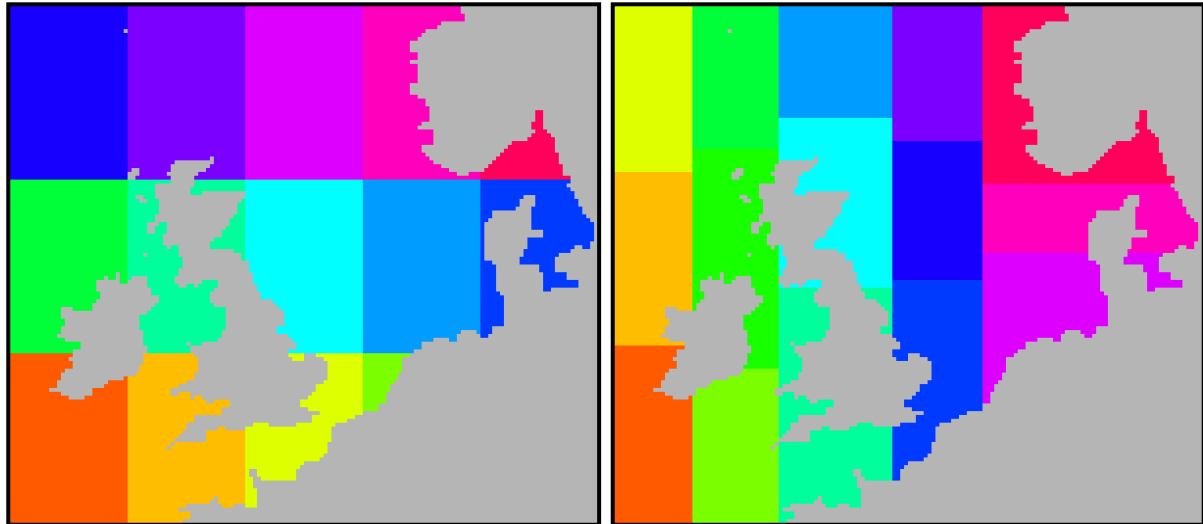


Figure 2: Comparison of simple (left) with recursive (right) partitioning for a model of the NW European shelf for 15 processors arranged in a 5x3 grid. The load balance improves from 62% to 97%.

We therefore factorise the number of processes for the run into numbers in the x- and y- directions, forming a rectangular process grid (see also [2]). The model domain is partitioned onto the same grid and one sub-domain mapped onto each process (we have considered allocating more than one sub-domain to each process but this has been rejected in current models as introducing additional complication without great advantage). The factorisation of the number of processes is usually chosen automatically to be as near square as possible. The reason for this is that the computational load for a process depends on the area of its sub-domain, whereas the communications load depends on the perimeter. Keeping the sub-domains close to square reduces the communications to computation ratio. The only situation where we may want to change this is if there is an advantage to having long, thin sub-domains, e.g. on a vector processor, and we retain the ability to specify manually at run-time the number of processes in x and y.

Simple two-dimensional partitioning of a typical land/sea distribution leads to a load imbalance due to the presence of land areas. We have addressed this problem by using a recursive bisection algorithm. In this algorithm we alternate between x- and y- directions. In each direction we count the sea grid points in each sub-domain and determine where to make a cut in the x- or y- direction which leaves equal (or as near equal as possible) numbers of sea points in each half. This process is repeated until the desired number of sub-domains is reached. For power-of-two numbers of sub-domains the algorithm proceeds as described. For other numbers we factorise the number into its prime factors and then each cut is not necessarily a bisection but a cutting into 3, 5, 7 etc. parts. Thus the algorithm may properly be described as a recursive k-section method. This is illustrated in figure 2 for a NW European shelf model running on 15 processes. The simple partitioning shown at the left has a load balance of just 62%, limiting performance to just 62% of ideal before starting with other overheads, and in fact we see that one process (at the bottom right) has no work at all to do, its sub-domain being entirely land. The recursive partition shown on the right improves the load balance to 97%. The remaining imbalance is due to the

restriction that we only form divisions between sub-domains along complete rows and columns.

The primary effect of the recursive partitioning is a better load balance. A secondary effect is additional complications to the communications structure. For example the process at the top-right of the partitions in figure 2, must in the recursive case send its western boundary data to two processes, whereas before it was only sent to one. However with closer consideration we realise that in the simple partitioning case all processes must send eight messages (two horizontal, two vertical and four corners), whereas for the recursive partitioning corner (diagonal) communications is only required in the rare case where four sub-domains meet. We have found that this reduces slightly the number of messages, and increases the average message length - a distinct advantage when running on high latency networks.

The partition over 256 processes shown in figure 3 illustrates a further problem. The process at the bottom right of the grid has a similar number of sea points to all the others but has a large amount of land and very long boundaries.

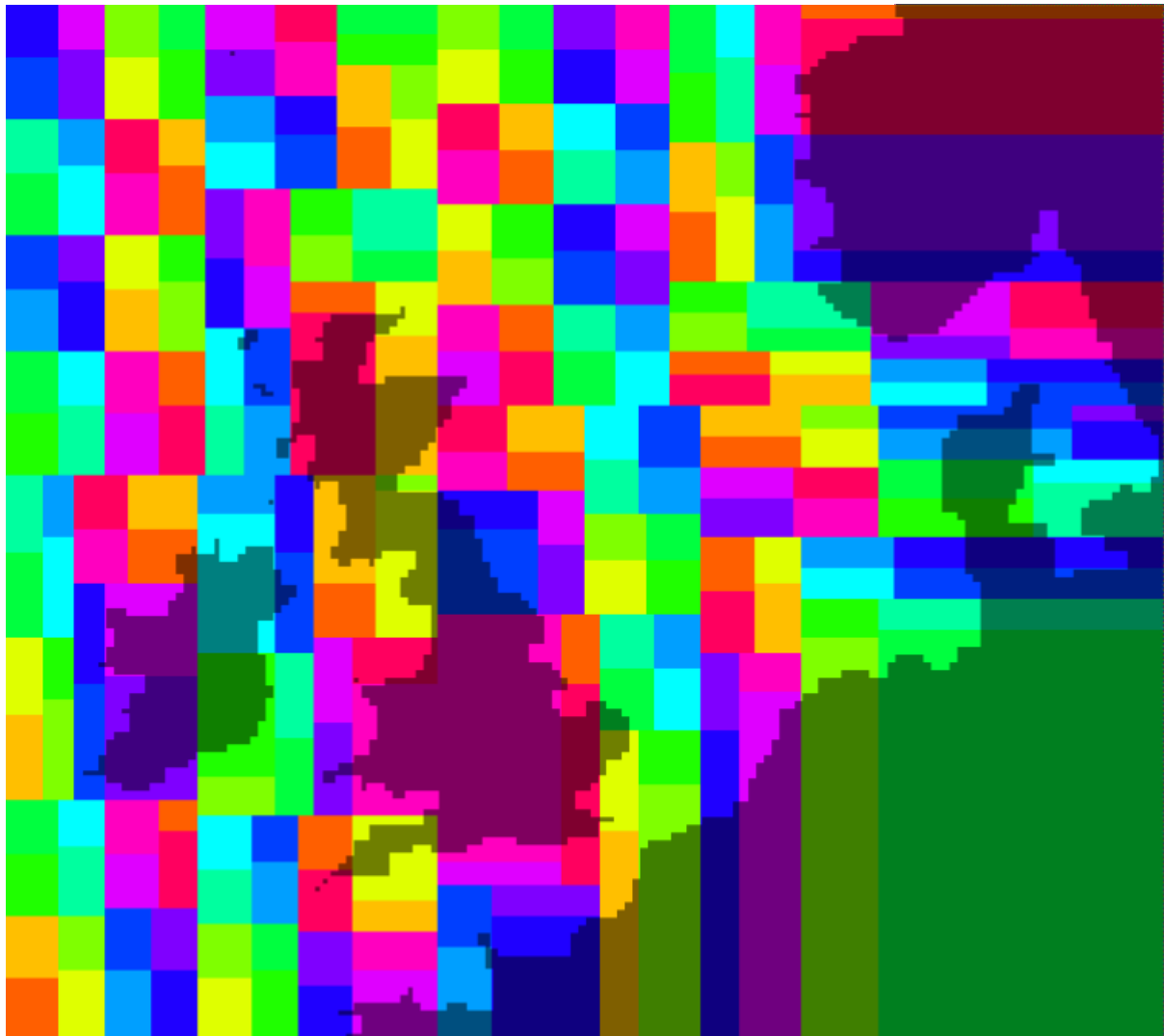


Figure 3: Recursive partitioning over 256 processes leaves the process at the bottom right of the grid with a large land area and long boundaries.

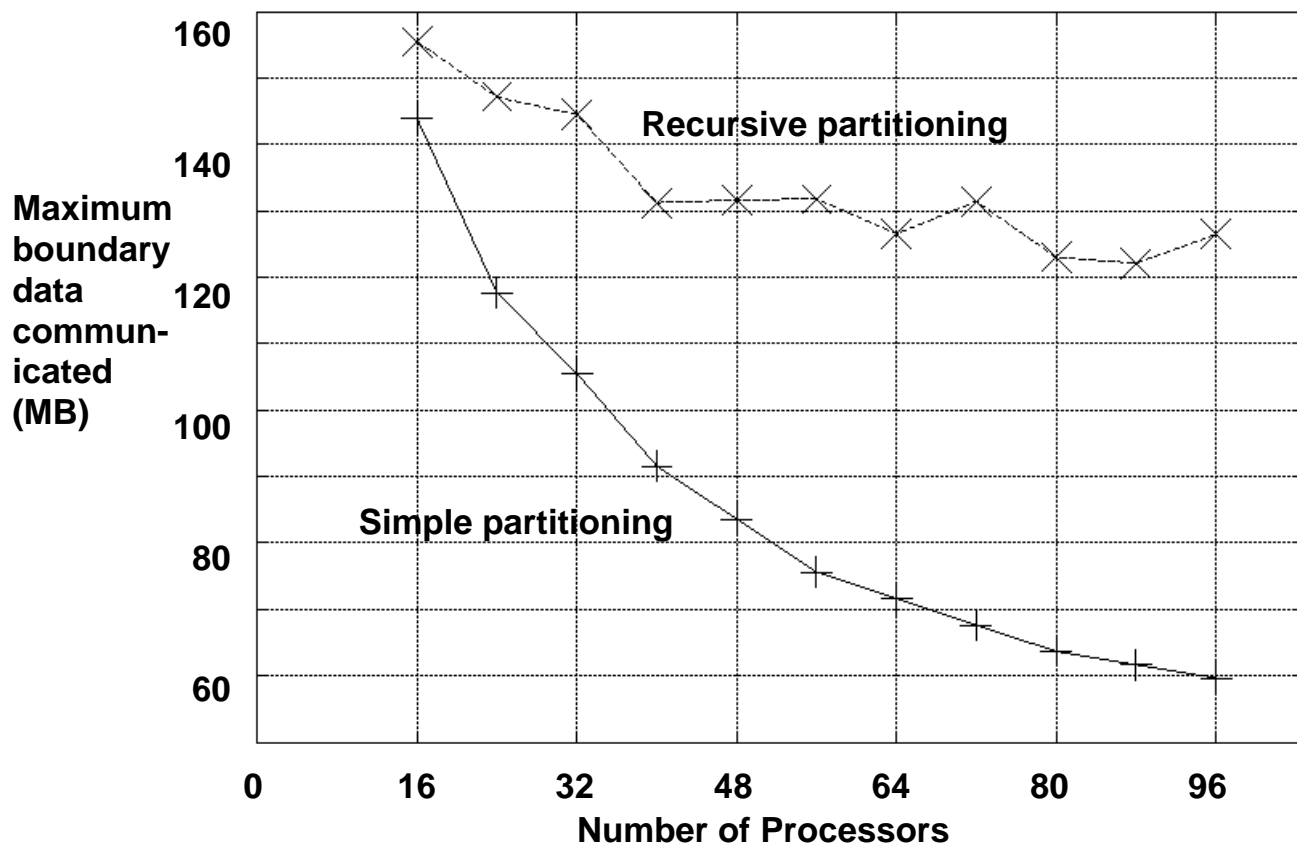


Figure 4: The maximum amount of data communicated across all the processors as a function of number of processors for the simple and recursive partitioning methods.

The initial implementation of the recursive partitioning algorithm did not take into account the long boundaries for sub-domains with many grid-points but only a few sea-points. This led to a communications load imbalance and poor scaling. Figure 4 shows the maximum amount of data communicated over all the processes for a range of processor numbers. Whereas in the simple case it scales down nicely (in common with the sub-domain size), for the recursive case it remains almost constant and the communications to computation ratio for this process increases. This is a case where we must look not only to the computational load balance but also to the communications load balance. In addition these large sub-domains included all the land in their memory allocation which would limit the ability of the code to scale to large domain sizes.

The solution is to add an extra step to the partitioning procedure which trims away rows and columns which are all land and which are surrounded by land and which therefore could take no part in the computation. This is shown in figure 5 for a 256 process partitioning of a larger area NW European shelf model. The black areas are removed from the model completely whereas the grey land areas remain.

Elsewhere authors have allowed a simple partitioning algorithm to over-allocate sub-domains in such a way that when sub-domains which are entirely land are removed, the required number remains [6, 7]. An alternative approach [3] uses sub-domains with an irregular shape.

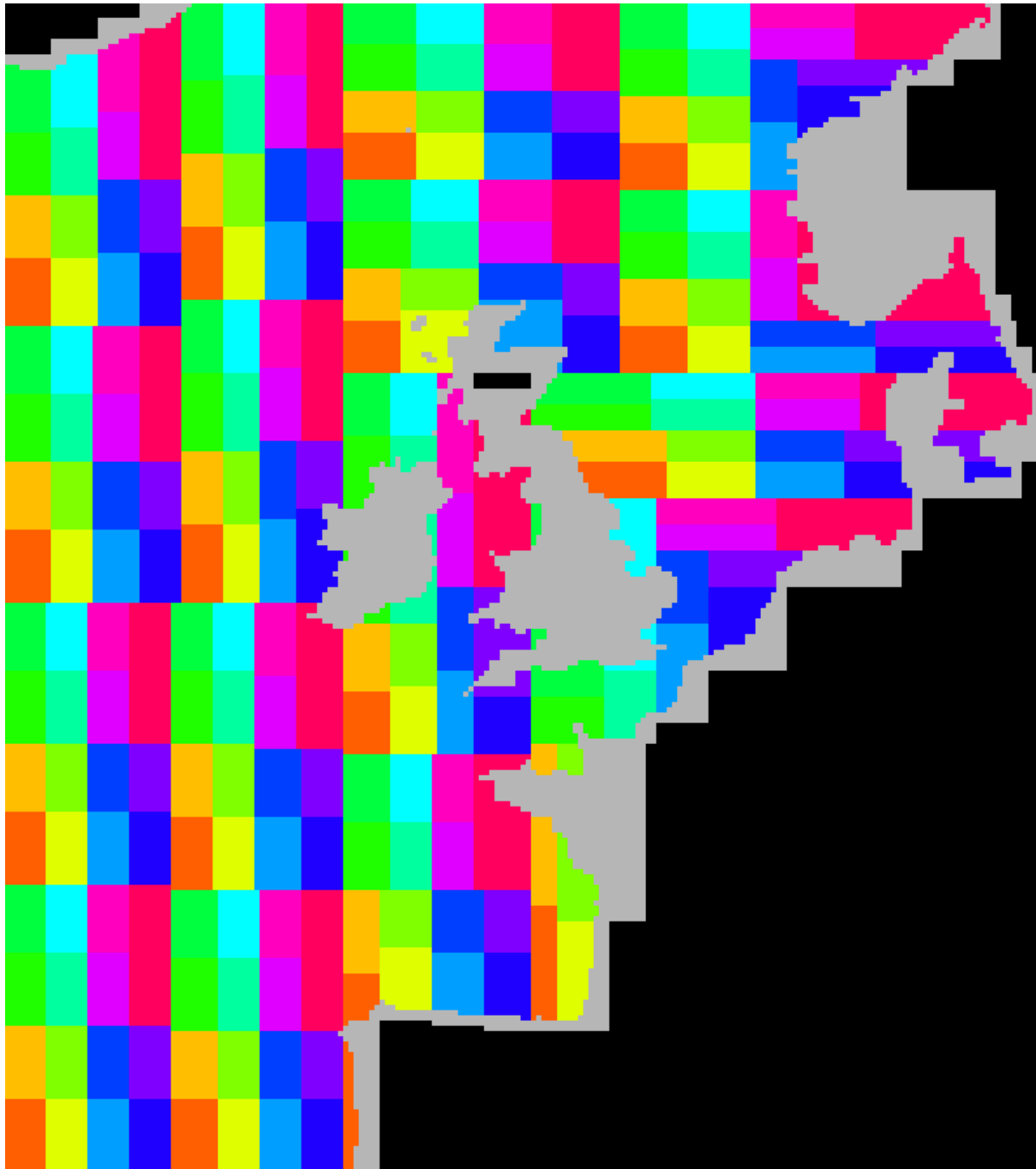


Figure 5: A recursive partition of a large-area NW European shelf model on 256 processors using the sub-domain trimming algorithm. Black areas are removed entirely from the model whereas grey land areas remain.

4 Boundary Exchange

We apply the *owner-computes* rule, whereby only the process which *owns* a data point is allowed to update it. This means that for a loop nest which has indices i and j running over the whole sub-domain, only terms such as $a(i,j)$ and not terms such as $a(i-1,j)$ are allowed on the left-hand side of assignments. Each process maintains a halo of points around its sub-domain. The halo contains a copy of boundary points *owned* by neighbouring sub-domains. When a process updates a field variable the

halo data in neighbouring sub-domains becomes out-of-date and must be refreshed by the exchange of boundary data before it is next used.

The finite difference calculations contain terms on the right-hand side of assignments such as $a(i-1,j)$ and $a(i-1,j+1)$ which access the halo data. Prior to such a calculation the programmer must ensure that the halo data is up-to-date. Calculations do not always contain references to all eight directions and the exchange routines allow one to specify precisely which directions are required.

For higher-order finite difference calculations there may be offsets greater than one in the finite difference stencil, e.g. $a(i-2,j)$ and $a(l,j+2)$. This is allowed for in the partitioning by allowing for halo widths greater than one. The interface to the exchange routines therefore include the width of the halo in storage and the width of the halo points which are to be communicated.

Separating the send and receive operations into different library calls allows the potential for overlapping computation with communication by placing some work between the send and receive. Whether this is effective in saving time depends on the implementation of the send and receive calls for a particular message passing environment and on the implementation of the message passing environment on the particular parallel platform.

In the initial design of the exchange routines, send and receive calls were hardwired for the eight exchange directions required for the regular partitioning. These are north, south, east and west for the edges, and north-east, north-west, south-east and south-west for the corner points. The introduction of the recursive partitioning led to a radical rewrite and generalisation. For recursive partitioning, sections of the edge boundary may be sent to different neighbouring processors. In the new design a communications list is constructed for sending and receiving boundary data. Each entry in the list contains the direction (coded for N, S, E, W, etc.), the id (e.g. the MPI rank) of the source or destination process, the x- and y-coordinates of the source data, the x- and y-coordinates of the destination data and the size of the data block in x and y. Knowing the location of *both* the source and destination data is not required for standard send-receive message passing, but allows us to use single-sided communications (e.g. SGI/Cray SHMEM or MPI-2) where this is available.

It is well known that sending all data and then receiving all data using MPI standard sends and receives is unsafe, as the MPI standard allows the sends to block until a receive has been posted, although on some systems where MPI is implemented differently this strategy will be successful. We therefore allow the option of using blocking or non-blocking communications. The solution to the deadlock problem described above using blocking communications is to send and receive in an odd-even fashion in each of the eight directions. That is, for the E-W direction, first processes with an odd x-processor coordinate send while processes with an even x-processor coordinate receive, and then the odds receive while the evens send. Next, for the N-S direction, processes with an odd y-processor coordinate send while processes with an even y-processor coordinate receive, etc. This serializes the communications into a number of phases (at most 16) but allows safe use of the synchronous communications mode.

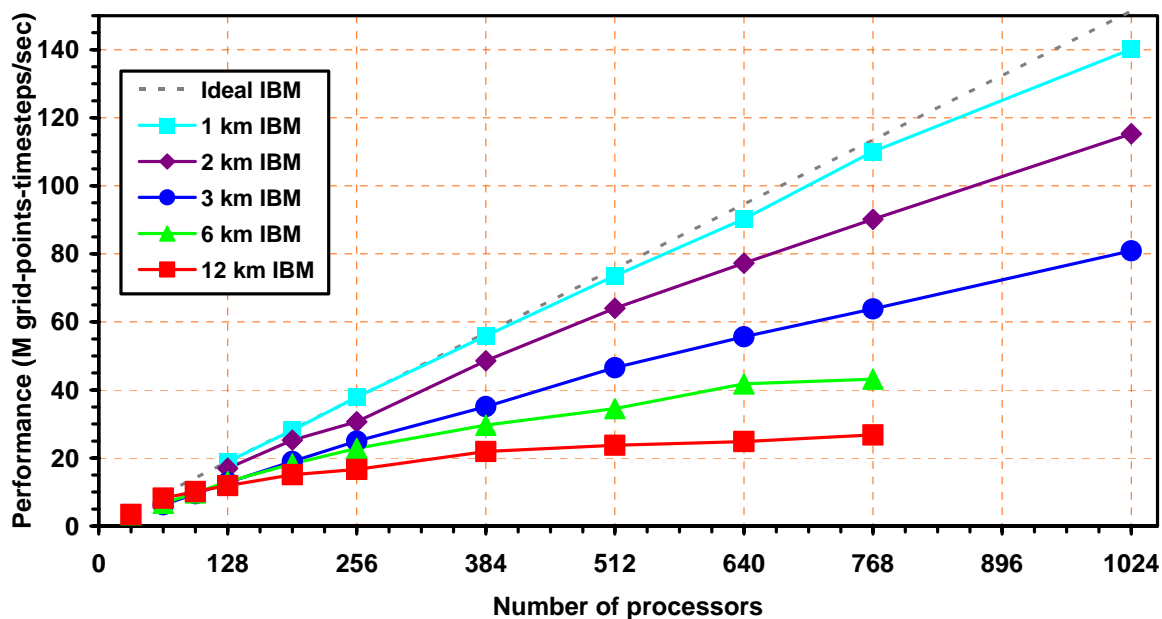


Figure 6: Performance of the POLCOMS hydrodynamic code at various resolutions on the HPCx phase1 system. The “Ideal” line shows a linear extrapolation from the 128 processor 1 km data point.

Much to be preferred on most systems is non-blocking communications, exemplified in MPI by the immediate communications mode. We routinely use this mode, in which all sends are posted using `MPI_issend`, all receives are posted with `MPI_irecv` and then a call to `MPI_waitall` ensures that all communications have been completed. This also allows us to overlap communications with calculations. Returning to the original code we note that boundary data exchange is required when a construct such as $a(i,j-1)$ appears on the right-hand side of an assignment. The receive part of the exchange must occur before this loop. The placement of the send call can be pushed back in time to the last time that the array a was updated i.e. appeared on the left-hand side of an assignment. There may be a considerable amount of calculation between these points allowing overlap of communications with calculations if that is facilitated by the implementation of the message passing library.

We note that an MPI communications call apparently designed for such exchange patterns (`MPI_sendrecv`) and which by design avoids the deadlock issue, is not appropriate in this case due to the anisotropic nature of the communications described above.

The implementation allows compile-time selection of MPI, PVM or SGI/Cray SHMEM communications libraries.

There is some global (collective) communications in the code, mainly in the distribution across all processes of input datasets read in on one process and in the subsequent gathering of data for output, but this is of minor importance and does not have a great impact on the overall performance.

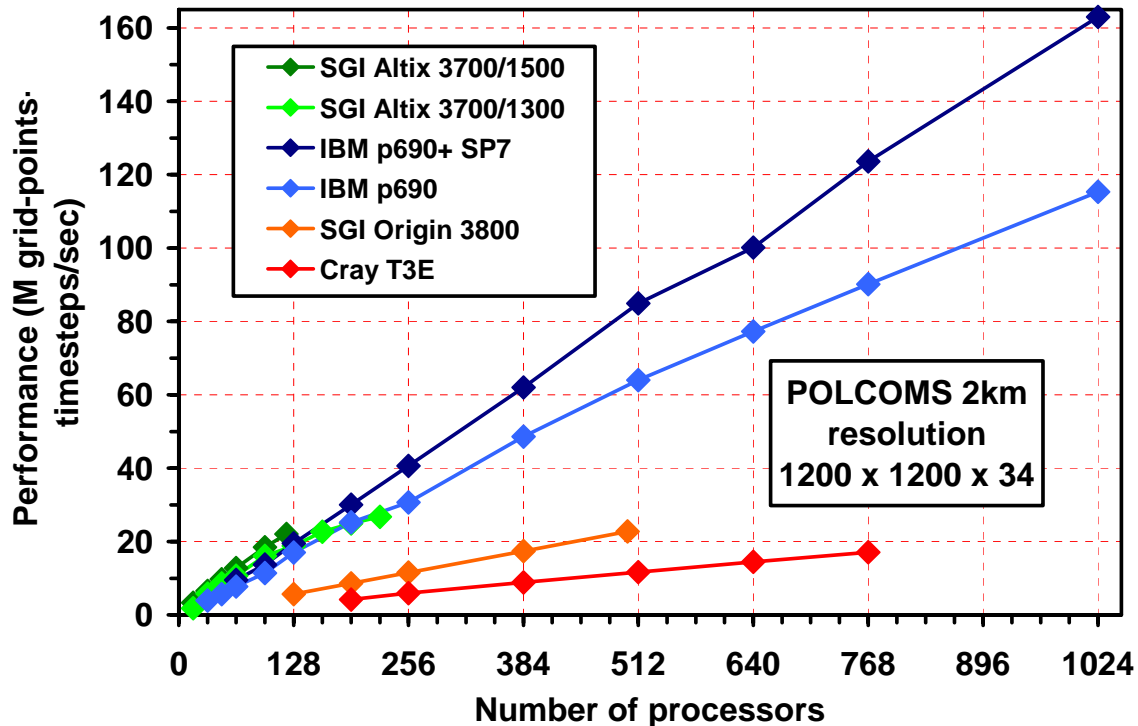


Figure 7: Performance of the POLCOMS hydrodynamic code at 2km resolution on the IBM p690/SP 1.3 GHz system, the SGI Origin 3800/R12k-400 and the Cray T3E/1200E

5 Terascaling the POLCOMS code

In order to assess the suitability of the POLCOMS hydrodynamic code for scaling to large domains at ultra-high resolution, we have designed a set of benchmarks which runs at grid sizes representative of resolutions from the current 12km down to 1km. The resolutions (and horizontal grid dimension) chosen are 12km (200), 6km (400), 3km (800), 2km (1200) and 1km (2400). The number of vertical levels was fixed at 34. In order to keep benchmark run times manageable, the runs were kept short (100 timesteps) and the initialisation and finishing times were subtracted from the total run time. In order correctly to compare runs with different grid sizes, performance is reported as the amount of work (gridpoints \times timesteps) divided by the time.

The systems used are the two phases of the HPCx system located at CCLRC Daresbury Laboratory and operated by the HPCx Consortium and a number of systems operated by CSAR at the University of Manchester, UK. HPCx phase1 was an IBM p690 cluster comprising 1.3GHz POWER4 processors and the SP2 "Colony" interconnect. This was superseded in May 2004 by the phase2 system with 1.7 GHz POWER4+ processors and IBM's High Performance Switch (HPS). Benchmark figures for phase2 include the performance critical upgrade to the switch software included in Service Pack 7. The CSAR systems are the Cray T3E/1200E ("turing"), removed from service in Dec 2003, the SGI Origin 3800/R12k-400 ("green") and the

SGI Altix 3700 system (“newton”). For the Altix there are results for the 1.3 GHz Itanium2 processors and using the latest partition containing 1.5 GHz processors.

Figure 6 shows how the performance varies on the HPCx phase1 system for increasing grid resolution. We find, as expected, that, as the grid size increases, the ratio of communication to computation in the code improves and so does the scalability with the higher resolution models scaling right out to 1024 processors.

In figure 7 we compare performance of the 2km resolution model between systems. At 2km resolution the code is scaling almost linearly on most systems

with the HPCx phase2 system delivering approx. 5.5 times the performance of the Cray T3E and 3.1 times the Origin 3800.

We have used the IBM *hpmcount* tool to analyse the performance of the code on the POWER4+ processors. This tool reports the cumulative values of a wide range of hardware counters, including instruction and operation counts, load and store issues and cache miss rates. For a 512-processor run at 2-km resolution, we record an average of 483.4 MegaFlop/s per processor implying that the 1024 processor run is achieving almost 0.5 TeraFlop/s. However impressive this may sound, we note that this performance level represents only 7.1% of the 6.8 GigaFlop/s peak performance of the POWER4+ processor. The usage of the fused multiply-add instruction (FMA) is at 48%, which should imply efficient use of the functional units. However we believe the overall performance is severely restricted by memory access issues and future work will concentrate on this area.

Despite the inefficient usage of processor hardware, we would like to conclude by noting that this performance level of the POLCOMS code allows previously unrealisable experiments to be carried out, pushing forward the capabilities in this area of computational science.

6 Summary

The key goal of this work on the POLCOMS code is to enable complex and increasingly coupled and multi-disciplinary scientific applications of the code to harness the potential performance of current and future high-performance parallel systems in order to meet critical scientific objectives. We have identified three key issues: management of the memory hierarchy, expression and management of concurrency and efficient sequential execution. These are key issues on all current systems.

We find that POLCOMS scales very well across all systems due to the successful optimisation of the memory access patterns for multi-level cache architectures and the maintenance of a high compute to communications ratio. We believe there is no substitute for careful design of data structures, data access (loop indexing) and data distribution. For legacy codes this may require a complete re-design.

References

- [1] Ashworth, M., Optimisation for Vector and RISC Processors, in *Towards Teracomputing*, eds. W. Zwiefelhofer and N. Kreitz, 1999, pp. 353-359, (World Scientific).
- [2] Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley Publishing, Reading, MA, 1995.
- [3] Gwilliam, C.S., The OCCAM Global Ocean Model <ftp://ftp.soc.soton.ac.uk/pub/occam/papers/ecmwf.ps.Z>, in *Coming of Age: Proceedings of the Sixth ECMWF Workshop on the use of Parallel Processors in Meteorology*, eds. G-R. Hoffmann and N. Kreitz, 1995, pp: 446-454 (World Scientific).
- [4] Holt, J.T., Proctor, R., Ashworth, M., Allen, J.I. and Blackford, J.C., Eddy Resolved Ecosystem Modelling in the Irish Sea, in *Realizing Teracomputing: Proceedings of the Tenth ECMWF Workshop on the Use of High Performance Computing in Meteorology*, eds. W. Zwiefelhofer and N. Kreitz, 2004, pp.268-278, (World Scientific).
- [5] Holt, J.T., Proctor, R., Blackford, J.C., Allen, J.I. and Ashworth, M., Advective controls on primary production in the stratified western Irish Sea: An eddy resolving model study, 2004, in press, *Journal of Geophysical Research*.
- [6] Konchady, M., Sood, A., Schopf, P.S., Implementation and performance evaluation of a parallel ocean model, 1998, *Parallel Computing*, 24, No. 2, pp. 181–203.
- [7] Wang, P., Ocean Modeling Grand Challenge Support, 1998, ESS Project FY98 Annual Report, NASA Jet Propulsion Laboratory <http://ess.jpl.nasa.gov/subpages/reports/98report/pw98two.html>

Publication history

An earlier version of this report was presented at the International Parallel and Distributed Processing Symposium 2004, 6th-8th April 2004, Santa Fe, New Mexico, USA.