A Performance Model of the Parallel Ocean Program

Darren J. Kerbyson
Performance and Architecture Laboratory (PAL)
Computer and Computational Sciences Division (CCS-3)
Los Alamos National Laboratory

Philip W. Jones
Theoretical Division (T-3)
Los Alamos National Laboratory

Proposed Running Head: POP Performance Model

Corresponding Author:

Philip W. Jones
Los Alamos National Laboratory
T-3, MS B216
P.O. Box 1663
Los Alamos, NM 87545-1663
Ph: 505-667-6387
Fax: 505-665-5926
pwjones@lanl.gov

Other authors:

Darren J. Kerbyson
Los Alamos National Laboratory
CCS-3, MS B256
P.O. Box 1663
Los Alamos, NM 87545-1663
Ph: 505-667-4913
Fax: 505-667-1126
djk@lanl.gov

SUMMARY

In this work we describe a performance model of the Parallel Ocean Program (POP). In particular the latest version of POP (v2.0) is considered which has similarities and differences to the earlier version (v1.4.3) as commonly used in climate simulations. The performance model encapsulates an understanding of POP's data decomposition, processing flow, and scaling characteristics. The model is parameterized in many of the main input parameters to POP as well as characteristics of a processing system such as network latency and bandwidth. The performance model has been validated to date on a medium sized (128 processor) AlphaServer ES40 system with the QsNet-1 interconnection network, and also on a larger scale (2,048 processor) Blue Gene/Light system. The accuracy of the performance model is high when using two standard benchmark configurations, one of which represents a realistic configuration similar to that used in Community Climate System Model coupled climate simulations. The performance model is also used to explore the performance of POP after possible optimizations to the code, and different task to processor assignment strategies, whose performance cannot be currently measured.

1 Introduction

The Parallel Ocean Program (POP) is an ocean general circulation model used for ocean and climate research. It is used in a variety of applications, including very high resolution eddy-resolving simulations of the ocean (Maltrud and McClean, 2005; Smith et al., 2000) and as the ocean component of coupled climate models like the Community Climate System Model (Blackmon et al., 2001). The combination of high resolution to resolve ocean eddies and long time scales required for climate and deep ocean circulation requires very high performance computing. The climate and ocean communities have access to a wide variety of such high performance computers with architectures ranging from clusters of commodity processors to vector machines like the Cray X1 and Japanese Earth Simulator. POP has therefore been designed to run efficiently across a wide variety of platforms and good performance across platforms has been demonstrated (Jones et al. 2004, Dunigan et al. 2003). In the most recent releases of POP, a flexible data decomposition scheme has been introduced to further increase the performance portability of the model.

The importance of analyzing, optimizing, and understanding the performance of large-scale applications such as POP increases as both systems and applications grow in size and in complexity. The performance of a system results from an interplay between the hardware architecture, the communication system, and the workload. Knowledge of the processor design, memory hierarchy, inter-processor and network system, and workload mapping is necessary in order to understand the factors that impact the achievable performance.

An important approach that provides insights into the achieved performance is that of performance modeling. Performance models can be constructed that encapsulate the key performance characteristics of a workload while also being parameterized in terms of the main characteristics of a machine. Performance models can be used for instance to examine the performance of systems without the need for extensive empirical analysis, to help identify performance bottlenecks, and provide input for tuning the application for a particular system. In addition, a performance model can be used to effectively replace many of the benchmarking activities for a machine of interest by providing reliable performance predictions for large-scale systems using measurements from only a small-scale system. Moreover, models can be used in situations which are not measurable – for instance in exploring the performance of design alternatives in future machines, or considering the impact of code changes in advance of implementing them.

The approach that we take in constructing a performance model is application centric. It involves understanding the processing flow in an application, the key data structures, how they use and are mapped to the available resources, and the effects of scaling. An analytical performance model of the application is constructed from this understanding. The aim is to keep the model of the application as general as possible but be parameterized in terms of the application's key characteristics. The model is based on a static analysis of the code and is parameterized in terms of the code’s dynamic behavior - i.e. those features which are not known through a static analysis.

We have had many successes using this performance modeling approach in the modeling of deterministic particle transport codes on structured (Hoisie et al. 2000), unstructured meshes (Kerbyson et al. 2003, Mathis and Kerbyson 2005), with non-deterministic particle transport (Mathis et al. 2005), and with hydro codes (Kerbyson et al. 2001). These performance models have been used in numerous performance studies, for example in the comparison of the Earth Simulator to other systems (Kerbyson et al. 2005), exploring possible future architectures (Kerbyson et al. 2002), and in the verification and performance optimization of ASCI Q at Los Alamos (Petrini et al. 2003).

There is a spectrum of different approaches to modeling the performance of systems and applications of which our application centric approach is only one. For example, trace data of both the sequential and parallel activity is often used to predict performance. An initial performance model of POP has been described using such an approach (Carrington et al. 2005). All of these modeling approaches can be used to help understand the measured performance, and allow certain what-if type performance questions to be analyzed across machines and configurations.

In this paper we first describe the POP model itself in Section 2, particularly those aspects of the model relevant to computational performance. In Section 3, we detail the performance model of POP which is validated against performance measurements on two systems in Section 4. In Section 5 we use the performance model to explore two performance scenarios which cannot be currently measured, namely the impact of code optimizations and in considering different task assignments to processors in a system.

2 Overview of POP

POP solves the primitive fluid equations on a sphere under the hydrostatic and Boussinesq approximations and is based on previous models of Bryan (1969), Cox (1984), Semtner (1986) and Chervin (1988) that use depth as the vertical coordinate. In the horizontal, POP supports any generalized orthogonal grid, including displaced-pole (Smith and Kortas, 1995) and tripole (Murray, 1996) grids that move the grid pole into land masses to avoid excessively small grid spacing near the pole singularity in latitude-longitude grids. Spatial derivatives are computed using second-order differencing on a staggered grid with velocities located at the logical northeast corner of tracer cells. Simulations begin from an initial state and are integrated forward in time using different methods for the baroclinic and barotropic modes. The fastest wave mode in the ocean is an external gravity wave mode, called the barotropic mode due to its uniform structure in the vertical. In POP, this two-dimensional barotropic mode is formulated as an elliptic equation for the surface pressure and is solved using a preconditioned conjugate gradient (PCG) solver (Dukowicz and Smith, 1994). The baroclinic portion explicitly integrates the three-dimensional fluid equations using a leapfrog scheme with periodic averaging steps to damp the leapfrog mode. Various physical parameterizations, subgrid models and other features are available (Smith and Gent, 2002).

POP can be run in either serial or parallel mode and can use thread-based (OpenMP) parallelism, message-passing (MPI) or a hybrid of the two. A flexible data decomposition scheme (described in Section 2.2 below) is used to decompose the horizontal grid; the vertical grid remains local. Halo regions or ghost cells are used to minimize communications by keeping local copies of nearest neighbor information for cells on the edges of a domain. The explicit three-dimensional baroclinic solver is the most computationally intensive and, with a halo depth of two, can be integrated with only a single ghost cell update of most prognostic fields. The preconditioned conjugate gradient solver for the barotropic mode consists of a two-dimensional nine-point stencil operator followed by global reductions necessary for the PCG iteration. The barotropic solver is therefore dominated by many small messages and relatively few floating point operations (flops).

2.1 Model configuration

For a given simulation, POP is configured using a combination of compile-time parameters and run-time inputs. Due to the difficulty of generating realistic bottom topography that also satisfies numerical constraints, a particular grid and bottom topography are used for many simulations. A few common configurations are made available for benchmarking purposes. The first is a *test* configuration that uses an internally generated grid and idealized topography. This configuration requires no input files beyond a simple configuration file and is ideally suited for initial benchmarking without needing to deal with I/O portability issues. A second configuration, called the *x1* (by one), is a one-degree (100km) resolution grid with realistic topography and the grid pole displaced into Greenland. It is configured as closely as possible to the way POP is used for long climate simulations. A third configuration, called the *x0.1*, is a very high resolution 1/10 degree (10km) grid configured as closely as possible to eddy-resolving simulations. In this work, we will only consider the first two configurations as the third requires very large computing resources

A wide variety of run-time inputs govern the choice of physical parameterizations and numerical methods. Here, we will only be concerned with a few main compile-time and run-time inputs that directly affect the computational performance of the model. These parameters primarily relate to the data decomposition as shown in Fig. 1 and Fig. 2. This is described below.

2.2 Data decomposition

Beginning with version 2.0 of POP, a flexible data decomposition scheme is used. For the horizontal grids supported in the model, the domain is logically-rectangular with a few special cases. The horizontal grid is therefore decomposed into two-dimensional blocks using a Cartesian decomposition. The block size is chosen by the user based on the particular machine architecture; blocks can be small for machines with small caches, but can be made large for vector processors. The size of a block is defined by the input parameters `block_size_x` and `block_size_y`. Thus the number of cells per block is `block_size_x × block_size_y × km`, where `km` is the depth, and the number of blocks is: `nx_global/block_size_x × ny_global/block_size_y` assuming that `block sizes` are both factors of the global horizontal dimensions `nx_global` and `ny_global` respectively (note that this latter factorization assumption is not required; POP will internally pad the domain when the block sizes are not factors of the horizontal dimensions.)

The arrangement of a block is shown in Fig. 1. Block(i,j) is shown along with it's four neighboring blocks, two in each dimension. The view is shown for the two horizontal dimensions and the third dimension (depth) extends into the page. Each block has a halo of ghost cells, permitting tasks to proceed independently with minimal communication and synchronization. The halo increases the size of a block to `(block_size_x + nghost×2) × (block_size_y + nghost×2) × km`, and also increases boundary surfaces between blocks.

Once the domain is decomposed into blocks, blocks with only land points are eliminated and the remaining blocks are distributed across processors. Multiple blocks may be assigned to each node, permitting both load balancing and hybrid parallelism with message passing between nodes and threading over multiple blocks within a processor. The distribution of blocks can be based on a static load balancing or can be a simple Cartesian distribution that simplifies communications. Different distributions can be chosen for the baroclinic and barotropic modes to permit an optimal distribution for the computationally-intensive baroclinic solver and the communication-intensive barotropic solver. Example block distributions are shown in Fig 2. Fig. 2a) shows an example global domain of 40 x 20 cells which are decomposed into blocks of size either 20 x 10 (Fig. 2b), or 5 x 5 (Fig. 2c). Blocks that contain only land are eliminated. A rake algorithm (Fonlupt et al. 1998) is used first in the X dimension, followed by the Y dimension to load-balance the blocks across the processors (Fig. 2d). Note that when one block is assigned to each processor, for example in Fig. 2b), the distribution is the same as used in the previous version of POP (v1.4.3), and no load-balancing is performed.

3 POP Performance Model

Our approach to modeling the performance of applications is to measure the single processor performance and to combine this with a model of the required parallel activities. The single processor performance can be either measured, for available systems, or provided as output from an architecture simulator in the case of a future system. There are two main parallel activities in POP:

Boundary exchanges – A stencil operator is used in the finite differencing in both the baroclinic and barotropic calculations. Boundaries in the horizontal dimensions need to be transferred between logical neighboring processors.

Global reductions – The PCG solver used in the barotropic calculation requires 2-3 global summations to be performed per solver iteration.

The communication and computation stages of POP are centered on a simulation *step*. A step can represent a varying amount of actual simulation time. Each involves a single call to both the baroclinic and barotropic calculations as well as a number of other operations. Note that the baroclinic and barotropic calculations constitute most of the run-time. We concern ourselves with modeling the performance of the run-time of POP – this can be subsequently used to determine the processing rate (simulation days per hour for example). One simplification we take in this work is to use the Cartesian distribution in which one block is assigned to each processor. This also has an advantage in that the performance model can also be directly applied to the earlier version of POP (v1.4.3). Modeling the full flexibility of block distributions introduces a broad parameter space in terms of block size, distribution of blocks and total workload. For example, smaller block sizes result in more land point elimination, resulting in a reduction in the total workload. The model may be extended to include the case of multiple blocks per processor but will require additional information specific to a particular problem configuration (including the actual number of blocks per processor, and details on the communicating processor-pairs used for the boundary exchanges). Despite the difficulties in modeling multiple blocks and flexible distribution schemes, the current model and the results below can provide some insight into performance using multiple blocks per processor; this will be discussed briefly in Section 4.1.

The run-time of POP, when considering only the baroclinic and barotropic calculations, can be described as:

$$T_{POP}(\mathbf{P}, \mathbf{N}, G) = T_{baroclinic}(\mathbf{P}, \mathbf{N}, G) + T_{barotropic}(\mathbf{P}, \mathbf{N}, G) \qquad (1)$$

where $\mathbf{P} = [P_x, P_y]$ is the number of processors in the two horizontal dimensions, $\mathbf{N} = [N_x, N_y, N_z]$ is used to denote the size of the global domain in the horizontal dimensions and depth respectively, and G is the number of ghost cells.

The run-time of the baroclinic is given by:

$$T_{baroclinic}(\mathbf{P}, \mathbf{N}, G) = T_{baroclinic\_comp}(\mathbf{P}, \mathbf{N}, G) + N_{bound\_clinic} \cdot T_{bound\_ex}(\mathbf{P}, \mathbf{N}, G) \qquad (2)$$

and the run-time of the barotropic is given by:

$$T_{barotropic}(\mathbf{P}, \mathbf{N}, G) = T_{barotropic\_comp}(\mathbf{P}, \mathbf{N}, G) + N_{bound\_tropic} \cdot T_{bound\_ex}(\mathbf{P}, \mathbf{N}, G) + N_{global\_sums} \cdot T_{global\_red}(\mathbf{P}) \qquad (3)$$

Both the baroclinic and barotropic times are separated into a sequential computation time, $T_{baroclinic\_comp}()$ and $T_{barotropic\_comp}()$, and a communication time which in the case of the barotropic includes both the boundary exchange time, $T_{bound\_ex}()$, and the global reduction time, $T_{global\_red}()$. The number of each of these operations is defined as N_{bound_clinic} and N_{bound_tropic} for the boundary exchanges, and N_{global_sums} for the global reductions. The computation and communication performances are described separately below.

3.1 Computation time

POP is commonly used in a strong-scaling processing mode in which parallelism is used to solve the same problem size in a reduced amount of time. In this case, the number of cells per processor decreases with increasing processor count. The time taken to process a cell is dependent on the parts of the memory hierarchy used and can vary. For instance a smaller problem per processor may utilize the memory cache to a greater extent than a larger problem.

The baroclinic compute time is modeled as

$$T_{baroclinic\_comp}(\mathbf{P},\mathbf{N},G) = Block\_size(\mathbf{P},\mathbf{N},G).T_{baroclinic\_cell}(Block\_size(\mathbf{P},\mathbf{N},G)) \tag{4}$$

and similarly the barotropic compute time is modeled as

$$T_{baroctropic\_comp}(\mathbf{P},\mathbf{N},G) = Block\_size(\mathbf{P},\mathbf{N},G).T_{baroctropic\_cell}(Block\_size(\mathbf{P},\mathbf{N},G)) \tag{5}$$

where the time per cell is given by $T_{baroclinic\_cell}()$, and $T_{barotropic\_cell}()$. The size of a block, $Block\_size(\mathbf{P},\mathbf{N},G)$, assuming one block per processor is given by

$$Block\_size(\mathbf{P},\mathbf{N},G) = N_z.\left\lceil \frac{N_x + 2.G}{P_x} \right\rceil . \left\lceil \frac{N_Y + 2.G}{P_Y} \right\rceil \tag{6}$$

For blocks that contain a mixture of land and ocean points, computations are normally performed on land points as performing extra computations is frequently less expensive than a conditional computation. *Block_size* is therefore a reasonable measure of the work performed in each block. Both the baroclinc and barotropic time per cell, $T_{baroclinic\_cell}()$, and $T_{barotropic\_cell}()$, need to be measured for a range of block sizes. Examples of these measurements are provided in Section 4 for two processing systems.

3.2 Parallel Activities

A boundary exchange is performed in two steps: one for the East-West (horizontal X dimension), and one for the North-South (horizontal Y dimension) exchange. Each exchange is done by two calls to MPI_Isend and two calls to MPI_Irecv followed by an MPI_Waitall. This is done to overlap any on-processor copies that may exist when multiple blocks are assigned to a processor; MPI is not used to perform such local copies. Note that the boundaries in either or both horizontal dimensions maybe defined as cyclic or closed. When using cyclic boundaries, the boundary on the lowest side of the global domain logically neighbors the highest side of the global domain.

From an execution of POP the number of boundary exchanges was found to be

$$N_{bound\_clinic} = ((nsteps - 1) \times 2 \times N_Z) \text{ , and } N_{bound\_tropic} = \left[nsteps \times \left[4 + Av\_scans \times (1 + \frac{1}{ncheck}) \right] \right] \quad (7)$$

for the baroclinic and barotropic calculations respectively. *nsteps* is the number of simulation steps. *Av_scans* is the average number of the PCG solver iterations per step, and *ncheck* is a further POP input which defines the number of iterations between a check for convergence of the PCG solver. Convergence of the PCG solver is specified by *solv_convrg*, the convergence criteria to be achieved, and *solv_max_iters* – the maximum number of iterations done if the convergence criteria is not met.

The time to perform a single boundary exchange is modeled as

$$T_{bound\_ex}(\mathbf{P}, \mathbf{N}, G) = T_{comm}(8.N_x.G, P_x.P_y, C_x) + T_{comm}(8.N_y(G+1), P_x.P_y, C_y) \quad (8)$$

where the boundary surfaces transferred in the X and Y dimensions are ($N_x.G$) and ($N_y.(G+1)$) eight-byte words respectively, and C_x, C_y represent the message contention for the X and Y boundary exchanges respectively. The message contention results from the assignment of the tasks onto a particular system network topology and impacts the bandwidth term of T_{comm}. It is discussed in Section 3.3. $T_{comm}(S,P,C)$ is the time taken to perform a bi-directional communication of size S bytes on a system of size P processors with a contention factor of C – the actual formulation of this is describe below.

The number of global summations in the barotropic calculation, each of a single word, was found to be

$$N_{global\_sum} = nsteps \times \left[1 + Av\_scans \times (2 + \frac{1}{ncheck}) \right] \quad (9)$$

The number of the parallel boundary exchanges and global summations, in equations 7 and 9, are valid for the inputs used in this work. The cost of performing a global summation can be modeled in a number of ways depending on its implementation. Here we consider it to be either measured for each processor count in a system, or is modeled as *log2(P)* stages in a binary tree reduction operation which is multiplied by 2 (since the operation is effectively a reduction followed by a broadcast).

A piece-wise linear model for the communication time is assumed which uses the latency, L_c, and bandwidth, B_c, of the communication network in the system. The effective communication latency and bandwidth vary depending on the size of a message and also the number of processors used (for instance when dealing with intra-node or inter-node communications for an SMP based machine).

$$T_{comm}(S,P,C) = L_c(S,P) + C.S.\frac{1}{B_c(S,P)} \tag{10}$$

The communication model utilizes the bandwidth and latencies of the communication network observed in a single direction when performing bi-directional communications, as is the case in POP for the boundary exchanges. They are obtained from a ping-pong type communication micro-benchmark which is independent of the application and in which the round-tip time when performing bi-directional communications is measured while varying the message size (typically increasing the message size in powers of 2). This should not be confused with the peak uni-directional communication performance of the network or peak measured bandwidths from a performance evaluation exercise.

3.3 Task Assignment

The arrangement of blocks across the processors in a system can significantly impact the performance of the boundary exchanges. When assuming a single block assigned to each processor, the logical arrangement of blocks is two-dimensional in a P_x and P_y array. However, the processors within a system may not be physically arranged in a two-dimensional array topology. We illustrate the assignment of tasks to two types of systems with different interconnection network topologies. The first is a cluster of SMP (Symmetric Multi-Processor) nodes interconnected with a fat-tree network, and the second is a three-dimensional torus network. These networks correspond to two types of high performance systems, including the ASCI Q machine at Los Alamos, and the Blue Gene/Light system recently introduced by IBM; the performance of both are analyzed further in Section 4. A three-dimensional torus network is also being used in the new Cray XT3 system, and the Raytheon TORO system.

An example fat-tree network is that implemented in the Quadrics QsNet-1 network (Petrini et al. 2002). This has been used in many AlphaServer systems in which the QsNet implements a quaternary fat-tree with each leaf of the tree connecting a 4-way SMP. Each processor within the SMP shares a single channel for communicating to other nodes in the system. In such cases when a boundary exchange is performed, contention for the communication channel will occur due to multiple boundaries being exchanged between processors on different nodes. This can be seen more clearly in the example shown in Fig. 3. The four processors in the central node will communicate boundaries to the four processors in each of the nodes to the North and to the South, and to only one processor in each of the nodes to the East and the West. When performing the North and South boundary exchanges a total of 8 messages will contend for a single communication channel, and when communicating East and West boundary exchanges only two messages will contend. The fat-tree offers a high degree of locality independence – the location of the neighboring nodes in the fat-tree is not important because the message communication performance is mostly insensitive to location. Note that different assignments of processes to processors may have different contention characteristics.

Conversely, the assignment ordering is important when a three-dimensional torus network is considered. For example, the best assignment of a logical two dimensional array into a three-dimensional torus will be where each X-Y plane of processors represents a rectangular sub-array of the original array. An example of this is shown in Fig. 4a). Alternate planes of the torus would be indexed in increasing and decreasing X coordinates, and increasing and decreasing Y coordinates. The indexing is shown on the outer row and columns of numbers in Fig. 4. This assignment results in no contention for any X-dimension boundary exchanges, and a small degree of contention for Y-dimension boundary exchanges (at the edges of the X-Y processor planes). Communications in the Z dimension of the torus are those that cross the dotted X-Y plane boundaries in Fig. 4a).

An alternate, simpler, three-dimensional torus assignment is one where processors are indexed in X, Y, Z ordering, and the first row of tasks in the logical two-dimensional array are just assigned to the first processors in the first X-Y plane, and so on as shown in Fig. 4b). We consider this is being simplier since the application process rank matches directly the processor ID in the C, Y, Z ordering. Here, contention occurs on most Y-dimension boundary exchanges. It can be seen that the exact layout of the communications can become quite complex, especially when adaptive routing in the network is used.

Note that the different assignment schemes, such as the optimum and simpler schemes above, are not under application control. They typically result from a system operation in which process ID's can be mapped to physical processors using a mapping function. In Blue Gene/L the notion of 'floor-plans' is used to describe this mapping function.

A summary of the communication contention for boundary exchanges is listed in Table 1 for processor counts up to 1,024 for a fat-tree with four-way SMP nodes, and a three-dimensional torus network. Two cases are shown for the torus, that using an optimum assignment, and that using the simpler X-Y-Z order assignment.

In the case of the fat-tree, the message contention does not increase when using 64 or more processors. In the case of the torus, the contention in the Y-dimension continues to increase in both cases. Note that the dimension of the torus is assumed to be 8 x 16 x 16. Only closed boundaries are considered in the contention factors in Table 1, a similar analysis for cyclic boundaries can be done.

3.4 Application Input Parameters

The performance model input parameters are based on the observations made using the test, and x1 inputs. The parameters representing the problem set-up that currently drive the performance model are listed in Table 2. Also listed are the values used for the test and x1 inputs. Note that

POP is typically used in a strong-scaling mode – that is that the overall spatial domain size, as defined in the input, remains constant no matter how many processors are used to solve it. This results in smaller sub-grid sizes (blocks) mapped onto a processor with increasing processor count. POP may also be used in a weak-scaling mode in which the overall problem size scales with the number of processors but this is not typical. The system input parameters, and their values, are discussed in Section 4.1.

4 Performance Model Validation

Before a performance model can be used, its prediction accuracy needs to be verified against observed performance on existing systems. In the validation presented here, two systems were used for measuring the performance of POP. The first is an AlphaServer Cluster containing 128 processors which is similar in architecture to the ASCI Q machine at Los Alamos. The second is a small version of the Blue Gene/Light system being built by IBM containing 2,048 nodes.

The AlphaServer cluster contained 32 four-way ES40 SMP nodes. Each processor was an Alpha EV68 with 8MB L2 cache running at 833MHz. The Alpha can issue two floating-point operations per cycle. The nodes are interconnected using the Quadrics QSnet-1 fat-tree network which has a large-message bandwidth of approximately 200MB/s, and a small message latency of 6µs in this system. Details of the Quadrics network are described by (Petrini et al. 2002).

Each Blue Gene/Light node consisted of a dual core embedded PowerPC 440 processor with a shared 4MB L3 cache running at 700MHz. Each core can issue four floating-point operations per cycle. Nodes are interconnected in a three-dimensional torus topology. Each communication channel in the network has a nearest-neighbor small message latency of 3.5µs and a large message bandwidth of 175MB/s. The testing below used only one of the processor cores per node (known as co-processor mode). Note that when using both processor cores, an increase in performance between a factor of 1.1 and 1.9 has been observed on other applications (Davis et al. 2004, and Almasi et al. 2004), but will depend on the impact of increased communication cost and the ratio of communication to computation in a particular configuration.

4.1 System Input Parameters

The single processor performance is an input to the POP performance model. The runtime of both the barotropic and baroclinic routines were measured while varying the size of the spatial domain. The time spent in the barotropic and baroclinic routines were recorded when using one processor per node, and in the case of the AlphaServer, when using all four processors in the node. Both cases are required for the model as contention for resources can occur when using all processors within a node which can significantly degrade the achieved performance. This contention is usually a result of having to share resources within the node and includes congestion on the memory buses.

Measured performance on both the AlphaServer and Blue Gene/Light machines is shown in Figure 5a) for the baroclinic, and in Figure 5b) for the barotropic. This is shown for the test input while varying the input spatial domain size. The baroclinic performance is shown in terms of time taken per cell per step. The barotropic performance is shown in terms of the time taken per cell per step per PCG iteration. All measurements are shown in terms of the processing time per processor. Note: the performance when using the x1 input has the same characteristic but was found to take a factor of 2.25 longer.

It can be noted that the time per cell increases when using all four processors in an AlphaServer node (mainly due to memory contention). The distinct regions in the curve are due to the memory hierarchy of the Alpha microprocessor. This can be seen clearly in the case of the baroclinic. A small problem size can fit into the L2 cache (left hand side of each curve), a large problem predominantly uses main memory (right hand side), and part cache and part main memory utilization occurs in between. This result can be used to estimate the optimal block size for this microprocessor and can restrict the block size range to explore when assigning multiple blocks per node in the full flexible decomposition strategy. Block size effects on land point elimination will be harder to model, though smaller blocks are always better for land point elimination. In addition, the block size is only part of the performance and modeling the full performance in the non-Cartesian case will also require an updated estimate of communication costs, given that some boundary exchanges will be local and others may occur between blocks that are no longer on neighbor processors in a load-balanced distribution. Such extensions to the performance model will be examined in future work. The same cache effect occurs in the barotropic case. The memory footprint used by barotropic is much less than that used by baroclinic. The transition from cache to main memory occurs at a far higher number of cells per processor (almost a factor of 100 higher).

As depicted in Figure 5, the single processor/single node performance can be modeled as a piece-wise linear curve. This is an approximation and can lead to an error in the input to the performance model. A summary of the system inputs to the performance model is given in Table 3. The piece-wise linear formulation of the communication model, in terms of L_c and B_c can be clearly seen for different ranges of the message size in Table 3. These are based on the measurements of bi-directional ping-pong microbenchmark as discussed earlier. Note that the time taken to perform a global reduction, $T_{global\_red}()$, was measured for each processor count on Blue Gene/L and was assumed to be *2*log(p)* times the uni-directional message latency on the AlphaServer.

Note that for each system that s modeled values for each of the system input parameters, as listed in Table 3, are required. For an existing system the computation costs can be obtained from executing POP using a range of global grid sizes on a single node, and the communication values obtained via micro-benchmarks. In the case of a future system, these values may be available from a simulator or be stated as the expected performance on such a system.

4.2 Measured and Predicted Performance

The measured and predicted performance of POP for the AlphaServer cluster using the test input is shown in Fig. 6a) and using the x1 input is shown in Fig. 6b). Note that markers indicate a measurement and a curve indicates model predictions. The time spent in baroclinic and barotropic is shown separately, and the total is simply the summation of the baroclinic and barotropic times. In a production simulation, there is some additional work outside of baroclinic and barotropic, including some I/O for forcing input and diagnostic ouput. These are generally a very small fraction (< 10%), so modeling only the baroclinic and barotropic portions of the code captures the bulk of the work and is the most predictable and reproducible measure. The measured and predicted performance of POP on the Blue Gene/L system is shown in Fig. 7 using the test input. No measurements have yet been made on the x1 input. The prediction accuracy is high in all cases.

It can be seen in Fig. 6 and Fig. 7 that the baroclinic dominates the run-time at low processor counts but scales well with increasing processor count. However, the barotropic performance scales up to approximately 512 processors on both the AlphaSever cluster, and the Blue Gene/L system and becomes the dominate component in the run-time. This is a consequence of the number of global summations required while the amount of work per cell performed is low. The scaling for both of these machines is actually reasonably good due to the good performance of both of the networks for collective operations.

Similarities can be seen in Figure 6 when using the x1 input. Prediction accuracy is again very high. In this case the barotropic is expected to scale (i.e. execution time decreases) to approximately 1,024 processors. A summary of the model prediction errors is given in Table 4. The maximum error across all the tests was 14 %, and a typical error is in the range 3-5%. Note that the errors result from an under-prediction in almost all cases. This is expected as the performance of POP includes the main characteristics of what the application is doing. It does not include external events, such as operating system kernel activities or daemons, which may negatively impact on the achieved performance. Such external events have been shown to have a significant impact on application performance on large-scale systems (Petrini et al. 2003). A single operating daemon occurring on a single node can delay the progress of the whole application if it occurs just prior to a global synchronization such as an allreduce in the barotropic phase of POP. This effect increases with system size. Systems such as the AlphaServer have a copy of almost the full operating system running on each node which can significantly impact performance. In contrast the Blue Gene/Light system has only a micro-kernel operating system which does not have many functions and no Daemons – this does not impact application performance significantly even at large processor counts.

5 Use of the Performance Model

Once the model has been validated, as detailed in Section 4, it may be used in a range of performance studies such as examining of the performance of possible future systems, or quantifying the impact on performance of changes to the application code. This is one of the main benefits of developing a performance model – to easily explore performance situations which cannot easily be measured. In this section we consider two performance questions of interest:

i) What will be the improvement in performance if the number of global summations was reduced to one per PCG iteration instead of the current two (e.g. Dongarra and Eijkhout, 2003; D'Azevedo, Eijkhout and Romine 1993) as is currently being introduced into POP by Bryan (private communication).

ii) What will be the improvement in performance if tasks are optimally assigned to processors in the BlueGene/L three-dimensional torus rather than the using the default X-Y-Z indexing?

The first question is answered though a slight modification to the model – namely replacing the two global summations per PCG iteration in Equation 9 to just one. The second question is answered by changing the message contention parameters, C_x and C_y, from the default X-Y-Z indexing values to the optimum values as listed in Table 1.

An analysis of both of these questions was undertaken. The performance improvement that may result from a change in the number of global summations is shown in Fig. 8a) for the AlphaServer system, and in Fig. 8b) for the Blue Gene/L system. The expected performance improvement increases with processor count. It reaches almost 30% on the barotopic time on the test input, and 20% on the x1 input on a 2,048 processor AlphaServer. On the Blue Gene/L system a performance improvement of 34% is expected on a processor count of 2,048. Note that the overall performance improvement is less than the improvement in just the barotropic time since the performance of the baroclinic remains unchanged.

The performance improvement that may arise from using an optimum assignment on an 8x16x16 Blue Gene/L system in shown in Fig. 9. Here the performance improvement mirrors the differences in the values of C_x and C_y as listed in Table 1 for the two assignment methods. When using 2,048 processors, the expected performance improvement of POP using the test input is 11%.

These two studies provide an insight into the possible performance improvements that can arise from both an optimization of the application code, as well as an optimization of the task assignment across a processing system. These performance studies can be readily undertaken by altering either the inputs to the performance model, or altering slightly the model formulation.

6 Conclusions

We have constructed a detailed performance model of POP which is very accurate across input and problem sizes. The model is parameterized in terms of the main inputs that are specified in the input of the actual application, and also in terms of the performance characteristics of the communication network as well as the assignment of tasks to processing nodes in the system.

Two inputs to POP have been used to examine the effectiveness of the model: the test input which is used for benchmarking purposes, and the degree 1 resolution model which is used in production runs in coupled climate simulations. Accuracy of the performance model is very good and matched closely measured performance on a 128 processor AlphaServer, and a 2,048 node Blue Gene/Light system for both of the main two routines of baroclinic and barotropic. Typical errors were found to be in the range 3-5% across a range of tests.

Once validated, the model can be used to explore a multitude of performance scenarios prior to the code being executed on the target system. Two such performance studies were undertaken in this work. The first was to consider the performance improvement that may arise if the number of global summations was halved from two to one per PCG iteration in the barotropic, and the second was to consider the difference in performance when using two alternative task assignment methods in the three-dimensional torus of Blue Gene/L.

We envisage using the performance model of POP here to compare performance across a range of systems for which the performance has not been measured. It will also find use in the development of new systems, to examine performance in advance of implementation, such as the systems being proposed and developed in the DARPA HPCS program. We also plan to use an extended performance model to examine optimal block size and block distributions for the POP 2.0 decomposition scheme.

Acknowledgements

This work was performed in part under the US Department of Energy Office of Science Climate Change Prediction Program and Scientific Discovery through Advanced Computing (SciDAC) program, in part under the US Department of Energy Office of Science contract, and in part under the DARPA HPCS contract. LANL is operated by the University of California for DOE under contract W-7405-ENG-36. We would also like to thank Bob Walkup and Jose Moirera for the access to the Blue Gene/Light system and for their expert support. This paper has been authored by a contractor of the U.S. Government under the above contract. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

REFERENCES

Almasi, G. et al. 2004. Unlocking the Performance of the BlueGene/L Supercomputer. in proc. IEEE/ACM Supercomputing, Pittsburgh, PA.

Blackmon, M., et al. 2001. The Community Climate System Model. Bull. Am. Meteorol. Soc. 82:2357–76.

Bryan K. 1969. A numerical method for the study of the circulation of the world ocean. J. Comput. Phys. 4:347.

Carrington, L., Snavelly, A., and Wolter, N. 2005. A Performance Prediction Framework for Scientific Applications. To appear in Future Generation Computer Systems, special issue on Large-Scale System Performance Modeling and Analysis, Elsevier.

Chervin, R.M. and A.J. Semtner 1988. An ocean modeling system for supercomputer architectures of the 1990s. In Proc. of the NATO Advanced Research Workshop on Climate-Ocean Interaction, edited by M. Schlesinger. Kluwer: Dordrecht.

Cox, M.D. 1984. A primitive equation, 3-dimensional model of the ocean. GFDL Ocean Group Technical Rept. No. 1, GFDL/NOAA: Princeton.

D'Azevedo, E.F., V.L. Eijkhout and C.H. Romine 1993. Conjugate gradient algorithms with reduced synchronization overhead on distributed memory multiprocessors. Computer Science Tech. Rep. CS-93-185, Univ. of Tennessee: Knoxville.

Dongarra, J. and V. Eijkhout 2003. Finite-choice algorithm optimization in conjugate gradients. Computer Science Tech. Rep. UT-CS-03-502, Univ. of Tennessee: Knoxville.

Dukowicz, J.K. and R.D. Smith 1994. Implicit free-surface method for the Bryan-Cox-Semtner ocean model. J. Geophys. Res. 99:7991–8014.

Dunnigan, T.H., Fahey, M.R., White, J.B., and Worley, P.H. 2003. Early Evaluation of the Cray X1. in proc. IEEE/ACM Supercomputing, Phoenix, AZ.

Hoisie, A., Lubeck, O., and Wasserman, H.J. 2000. Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications. Int. J. of High Performance Computing Applications, Sage Science Press, 14(4):330-346.

Fonlupt, C., P. Marquet and J.-L. Dekeyser 1998. Data-parallel load balancing strategies. Parallel Computing, 24:1665-1684

Jones, P.W., P.H. Worley, Y. Yoshida, J.B. White III, and J. Levesque 2004. Practical performance portability in the Parallel Ocean Program (POP). Concurrency and Computation: Practice and Experience, in press.

Kerbyson D.J., Alme, H.J., Hoisie, A., Petrini, F., Wasserman, H.J., Gittings, M. 2001. Predictive Performance and Scalability Modeling of a Large-Scale Application. In proc. of IEEE/ACM Supercomputing, Denver, Co.

Kerbyson, D.J, Hoisie, A and Wasserman, H.J. 2002. Exploring Advanced Architectures using Performance Prediction. In Innovative Architecture for Future Generation High-Performance Processors and Systems, IEEE Computer Society Press, pp. 27-37.

Kerbyson, D.J, Hoisie, A and Pautz, S. 2003. Performance Modeling of Deterministic Transport Computations. In Performance Analysis and Grid Computing, V. Getov, M. Gerndt, A. Hoisie, A. Malony, and B. Miller (eds), Kluwer, pp. 21-39.

Kerbyson, D.J., Hoisie, A., and Wasserman, H.J. 2005. A Performance Comparison between the Earth Simulator and other Terascale Systems on a Characteristic ASCI Workload. To appear in Concurrency and Computation: Practice and Experience.

Mathis, M.M., and Kerbyson, D.J. 2005. Performance Modeling of Unstructured Mesh Particle Transport Computations. To appear in Int. J. of Supercomputing.

Mathis, M.M., Kerbyson, D.J., and Hoisie, A. 2005. A Performance Model of non-Deterministic Particle Transport on Large-Scale Systems. To appear in Future Generation Computer Systems, special issue on Large-Scale System Performance Modeling and Analysis, Elsevier.

Maltrud, M.E. and J.L. McClean 2005. An eddy-resolving global 1/10 degree ocean simulation, Ocean Modelling 8:31-54.

Murray, R.J. 1996. Explicit generation of orthogonal grids for ocean models. J. Comp. Phys. 126:251.

Petrini, F., Feng, W.C., Hoisie, A., Coll, S., and Frachtenberg, E. 2002. The Quadrics network: High-performance clustering technology. IEEE Micro, 22(1):46–57.

Petrini, F., Kerbyson, D.J., and Pakin, S 2003. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. in proc. IEEE/ACM Supercomputing, Phoenix. Awarded best paper.

Semtner Jr., A.J. 1986. Finite-difference formulation of a world ocean model. In Advanced Physical Oceanographic Numerical Modeling, edited by J.J. O'Brien Reidel: Dordrecht.

Smith, R.D. and P. Gent 2002. Reference manual for the Parallel Ocean Program (POP). Los Alamos Unclassified Report LA-UR-02-2484.

Smith, R.D. and S. Kortas 1995 Curvilinear coordinates for global ocean models. Los Alamos Unclassified Report LA-UR-95-1146.

Smith, R.D., M.E. Maltrud, F.O. Bryan and M.W. Hecht 2000. Numerical simulation of the North Atlantic ocean at 1/10 degree. J. Phys. Oceanogr. 30:1532–61.

List of Table captions

Processor Count			Fat-tree 4-way SMP		3-D Torus (8x16x16) Optimum		3-D Torus (8x16x16) Simple	
P	P_x	P_y	C_x	C_y	C_x	C_y	C_x	C_y
2	2	1	1	0	1	1	1	1
4	2	2	2	2	1	1	1	2
8	4	2	2	4	1	1	1	2
16	4	4	2	8	1	1	1	2
32	8	4	2	8	1	1	1	1
64	8	8	2	8	1	1	1	1
128	16	8	2	8	1	1	1	2
256	16	16	2	8	1	1	1	2
512	32	16	2	8	1	2	1	4
1024	32	32	2	8	1	2	1	4
2048	64	32	2	8	1	4	1	8
4096	64	64	2	8	1	4	1	8

Table 1

Parameter	test	x1	Description
N_x (*nx_global*)	192	320	Overall spatial domain size in X
N_y (*ny_global*)	128	384	Overall spatial domain size in Y
N_z (*km*)	20	40	Number of layers (depth) in the spatial grid
G (*nghost*)	2	2	Number of ghost cells
nsteps	20	50	Number of simulation steps
solv_ncheck	10	10	Number of PCG iterations between convergence checks
Av_nscans	69	179	Average number of PCG iterations per step

Table 2.

Parameter	AlphaServer	Blue Gene/L	Description
P	1..4096	1..4096	processor count
P_{node}	4	1	processors per node
$T_{baroclinic\_cell}(E)$	P < 4 (μs) $\begin{cases} 5.0 & E <= 11K \\ 3.0Ln(E) - 22.3 & 11K < E < 250K \\ 11.3 & E > 250K \end{cases}$ P >= 4 (μs) $\begin{cases} 8.1 & E <= 11K \\ 4.1Ln(E) - 28.8 & 11K < E < 250K \\ 21.0 & E > 250K \end{cases}$	(μs) $1.96+0.2Ln(E)$	baroclinic computation time (per cell)
$T_{barotropic\_cell}(E)$	P < 4 (ns) $\begin{cases} 5.5 & E <= 1M \\ 3.5Ln(E) - 4.2 & E > 1M \end{cases}$ P >= 4 (ns) $\begin{cases} 6.5 & E <= 1M \\ 7.3Ln(E) - 9.5 & E > 1M \end{cases}$	15 ns	barotropic computation time (per cell per PCG iteration)
$L_c(S,P)$	P <= 4 (μs) $\begin{cases} 15.0 & S <= 32 \\ 15.0 & 32 < S < 512 \\ 27.2 & S >= 512 \end{cases}$ P > 4 (μs) $\begin{cases} 11.0 & S <= 64 \\ 9.7 & 64 < S < 512 \\ 14.5 & S >= 512 \end{cases}$	(μs) $\begin{cases} 4.15 & S <= 32 \\ 3.91 & 32 < S < 512 \\ 7.46 & S >= 512 \end{cases}$	MPI latency
$B_c(S,P)$	P <= 4 (ns) $\begin{cases} 0 & S <= 32 \\ 25.6 & 32 < S < 512 \\ 3.3 & S >= 512 \end{cases}$ P > 4 (ns) $\begin{cases} 0 & S <= 64 \\ 27.3 & 64 < S < 512 \\ 12.7 & S >= 512 \end{cases}$	(ns) $\begin{cases} 6.3 & S <= 32 \\ 12.1 & 32 < S < 512 \\ 6.5 & S >= 512 \end{cases}$	MPI time per byte

Table 3.

	AlphaServer (test)		AlphaServer (x1)		Blue Gene/L (test)	
	max.	avg.	max.	avg.	max.	avg.
baroclinic	13.3 %	3.9 %	4.6 %	2.0 %	10.7 %	5.4 %
barotropic	7.5 %	3.7 %	12.7 %	4.3 %	14.0 %	4.3 %
total	11.2 %	3.4 %	4.7 %	2.0 %	10.1 %	4.0 %

Table 4.

List of Figure captions:

Figure 1. A block in the two-dimensional data decomposition of POP.

Figure 2. Example data decomposition scheme in POP showing (a) original global domain, (b) Cartesian assignment for a block size of 10 x 20, (c) decomposition using 5 x 5 blocks, and (d) distribution of 5 x 5 blocks to four processors.

Figure 3. Boundary exchanges required between processors on a four-way SMP node.

Figure 4. Assignment of an 8 x 8 two-dimensional array of tasks to a 4 x 4 x 4 three-dimensional torus of processors. (a) Optimum assignment in which each X-Y plane contains a rectangular sub-array of the total task array. (b) Non-optimum assignment using a default X-Y-Z processor indexing.

Figure 5. POP processing time on one and four processors of the AlphaServer, and one processor of Blue Gene/L as a function of the problem size per processor. (a) time per cell in baroclinic, and (b) time per cell per PCG iteration in barotropic.

Figure 6. Measured and predicted runtime of POP on the AlphaServer cluster. (a) using the test input, and (b) using the x1 input..

Figure 7. Measured and predicted runtime of POP on the Blue Gene/L system using the test input.

Figure 8. Expected performance improvements on the barotropic and total runtime resulting from a reduced number of global summations per PCG iteration. (a) AlphaServer for both test and x1 inputs, and (b) Blue Gene/L for the test input.

Figure 9. Expected performance improvements on the baroclinic, barotropic and total runtime on Blue Gene/L resulting from using an optimum three-dimensional torus assignment for the test input.

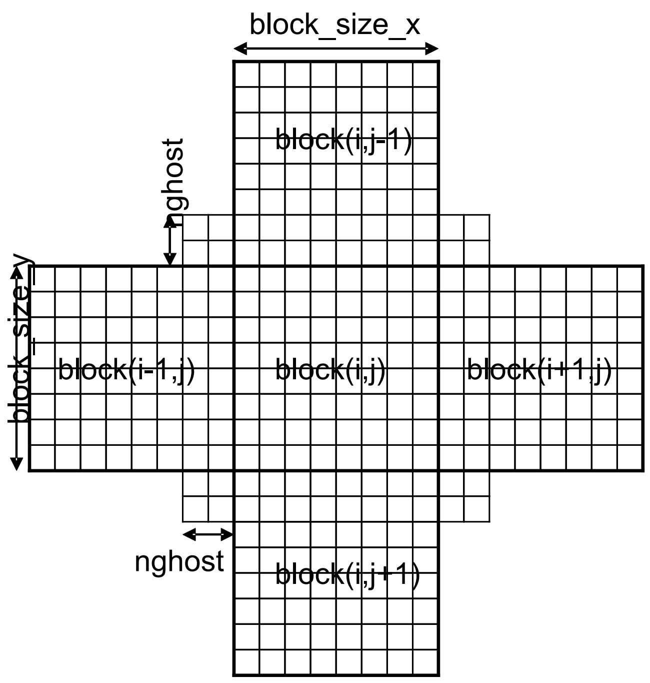
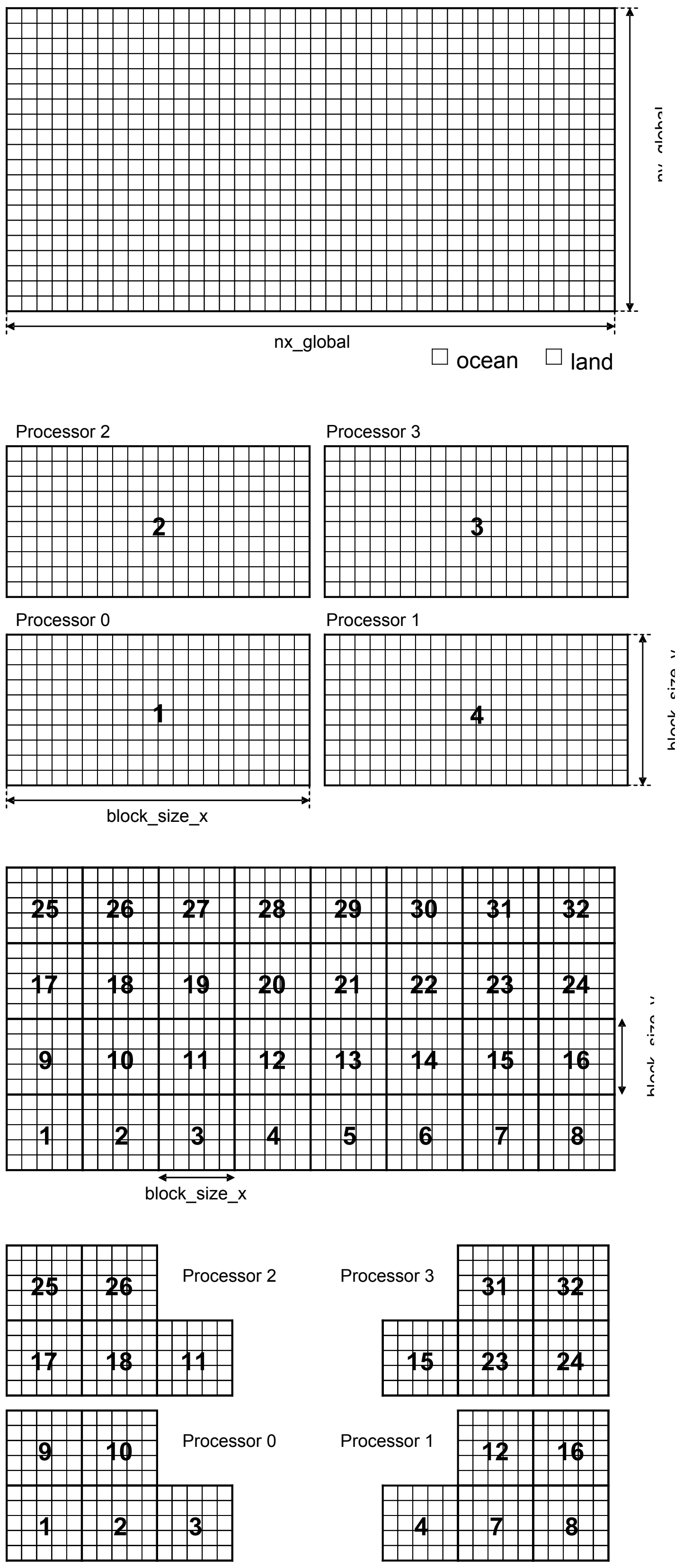

block_size_x
block(i,j-1)
nghost
block_size_y
block(i-1,j)
block(i,j)
block(i+1,j)
nghost
block(i,j+1)

Figure 1.



nx_global
☐ ocean ☐ land
(a)
Processor 2
Processor 3
2
3
Processor 0
Processor 1
1
4
block_size_x
(b)
25 26 27 28 29 30 31 32
17 18 19 20 21 22 23 24
9 10 11 12 13 14 15 16
1 2 3 4 5 6 7 8
block_size_x
(c)
Processor 2
Processor 3
25 26 17 18 11
31 32 15 23 24
Processor 0
Processor 1
9 10 1 2 3
12 16 4 7 8
(d)

Figure 2.

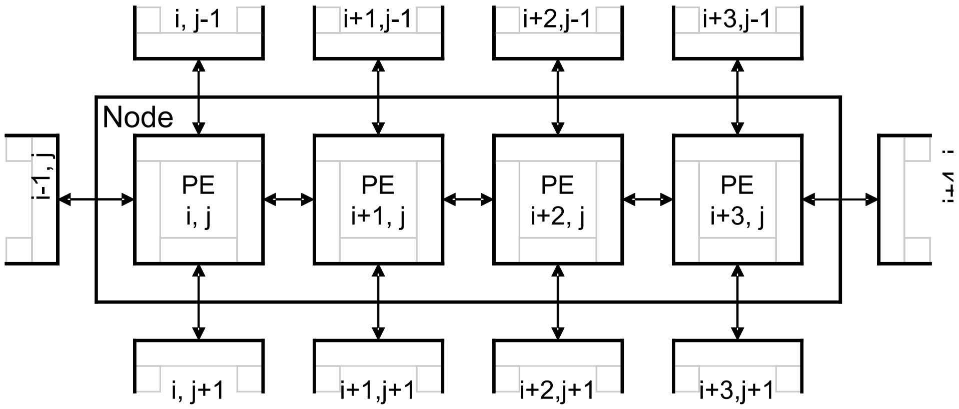
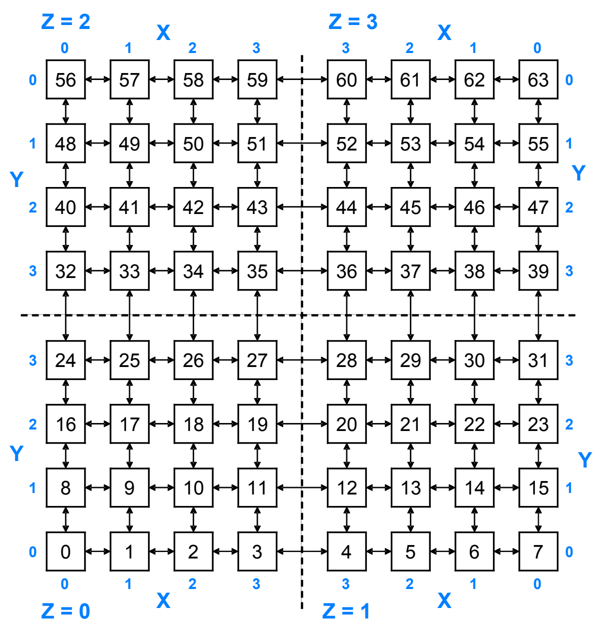

i, j-1
i+1,j-1
i+2,j-1
i+3,j-1
Node
i-1, j
PE
i, j
PE
i+1, j
PE
i+2, j
PE
i+3, j
i, j+1
i+1,j+1
i+2,j+1
i+3,j+1

Figure 3.



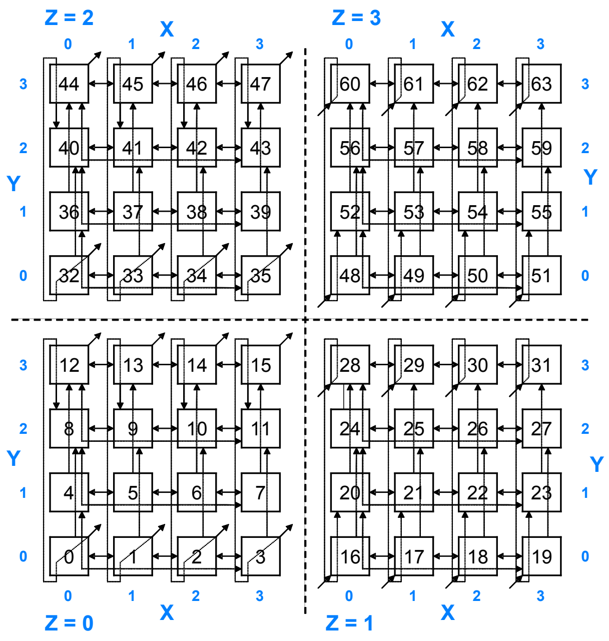
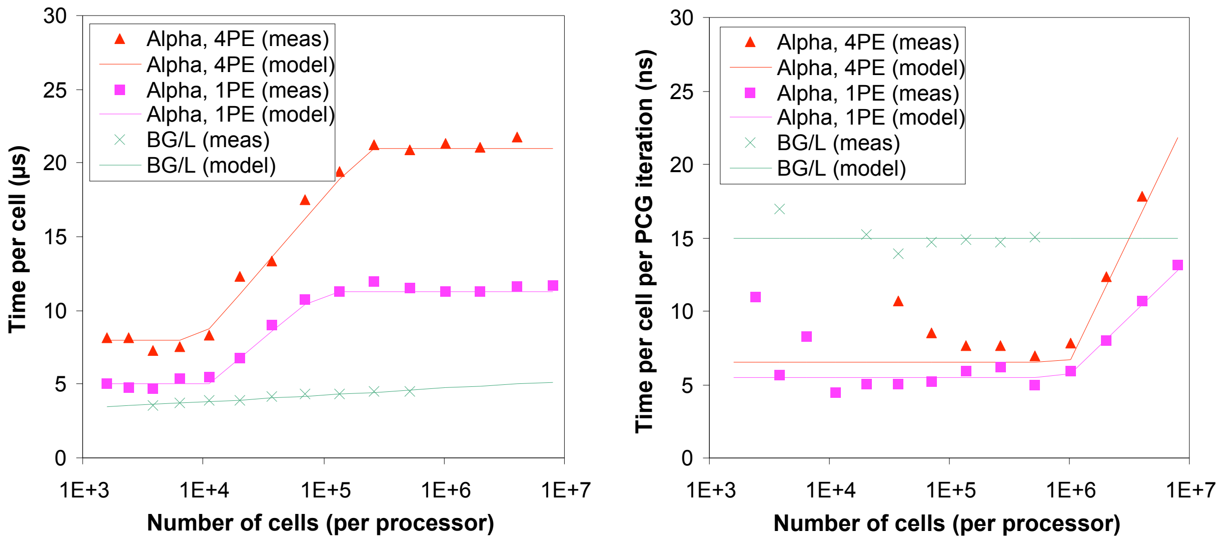
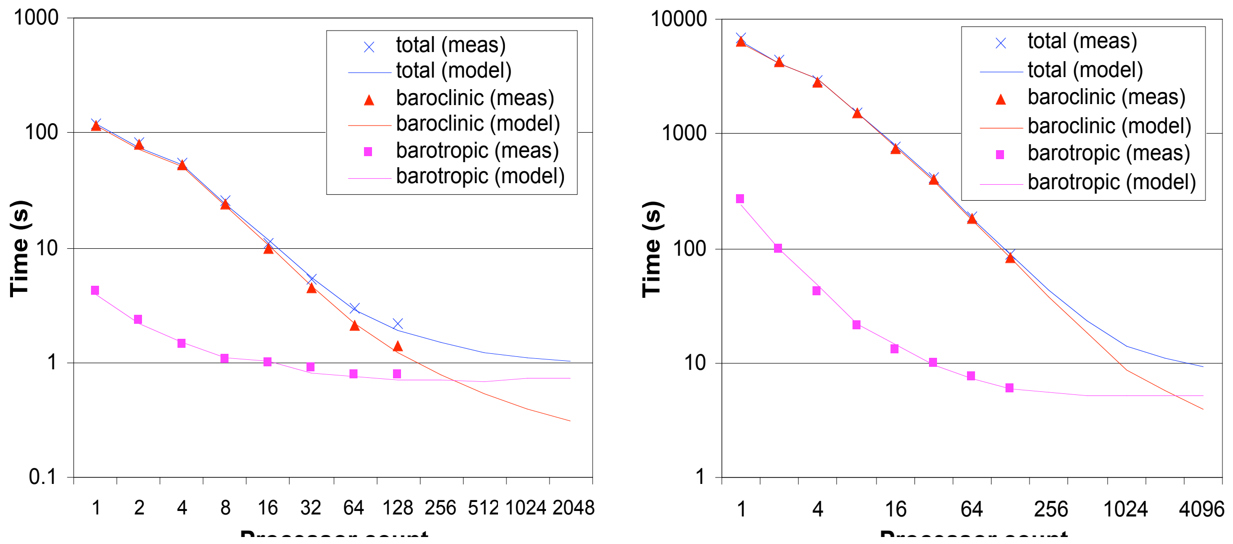


Figure 4.



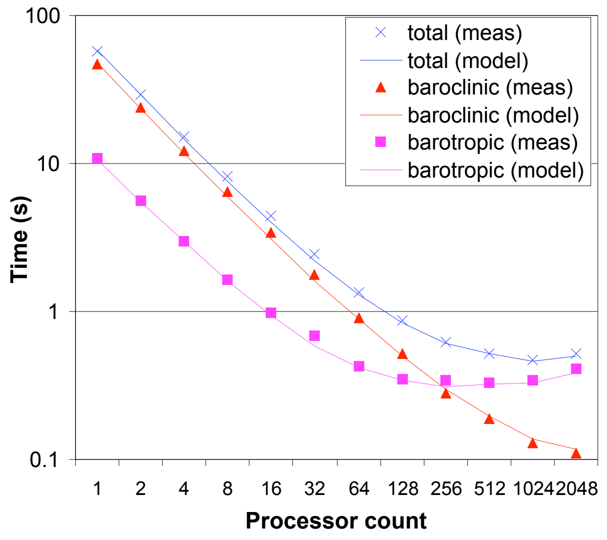
Alpha, 4PE (meas)
Alpha, 4PE (model)
Alpha, 1PE (meas)
Alpha, 1PE (model)
BG/L (meas)
BG/L (model)
Time per cell (µs)
Number of cells (per processor)
Time per cell per PCG iteration (ns)
Number of cells (per processor)
0
5
10
15
20
25
30
1E+3
1E+4
1E+5
1E+6
1E+7

Figure 5



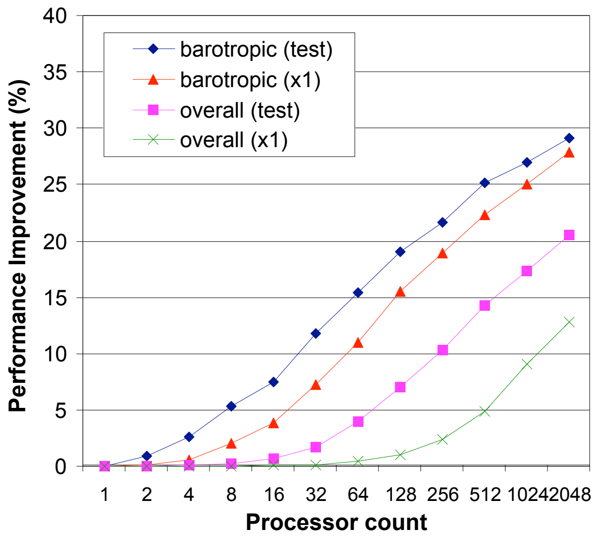
1000
100
10
1
0.1
Time (s)
total (meas)
total (model)
baroclinic (meas)
baroclinic (model)
barotropic (meas)
barotropic (model)
1 2 4 8 16 32 64 128 256 512 1024 2048
10000
1000
100
10
1
Time (s)
1 4 16 64 256 1024 4096

Figure 6.

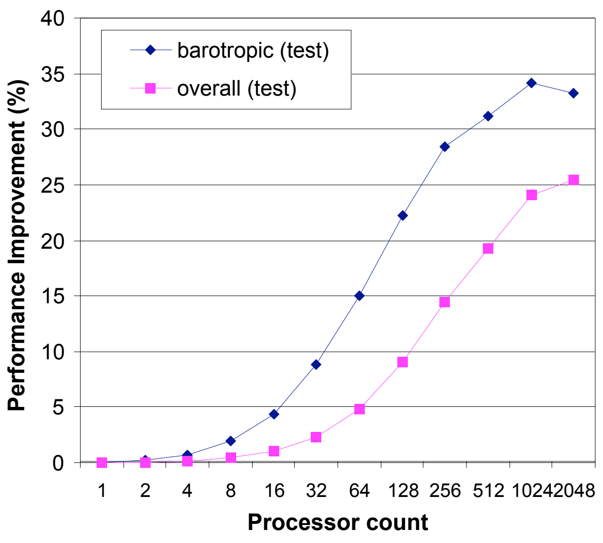


100
10
1
0.1
Time (s)
1
2
4
8
16
32
64
128
256
512
1024
2048
Processor count
total (meas)
total (model)
baroclinic (meas)
baroclinic (model)
barotropic (meas)
barotropic (model)

Figure 7.



Performance Improvement (%)
Processor count
barotropic (test)
barotropic (x1)
overall (test)
overall (x1)
0
5
10
15
20
25
30
35
40
1
2
4
8
16
32
64
128
256
512
1024
2048



Performance Improvement (%)
Processor count
barotropic (test)
overall (test)
0
5
10
15
20
25
30
35
40
1
2
4
8
16
32
64
128
256
512
1024
2048

Figure 8.

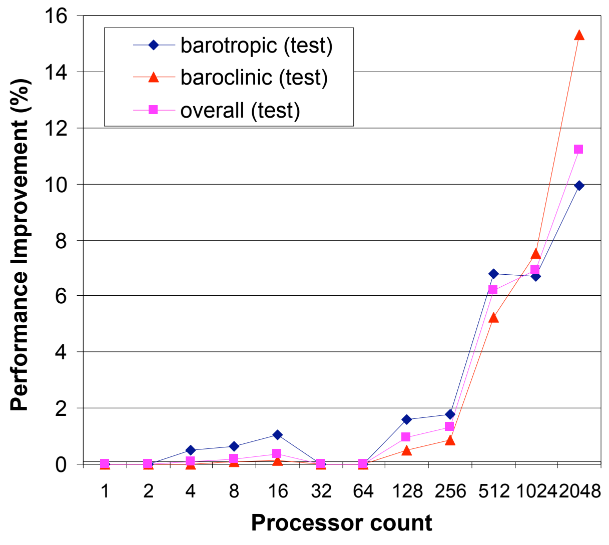

barotropic (test)
baroclinic (test)
overall (test)
Performance Improvement (%)
Processor count
0
2
4
6
8
10
12
14
16
1
2
4
8
16
32
64
128
256
512
1024
2048

Figure 9.