Inverse Space-Filling Curve Partitioning of a Global Ocean Model

John M. Dennis[1]

[1]Computational & Information Systems Laboratory
National Center for Atmospheric Research
Boulder, CO 80307-3000
dennis@ucar.edu

Abstract

In this paper, we describe how inverse space-filling curve partitioning is used to increase the simulation rate of a global ocean model. Space-filling curve partitioning allows for the elimination of load imbalance in the computational grid due to land points. Improved load balance combined with code modifications within the conjugate gradient solver significantly increase the simulation rate of the Parallel Ocean Program at high resolution. The simulation rate for a high resolution model nearly doubled from 4.0 to 7.9 simulated years per day on 28,972 IBM Blue Gene/L processors. We also demonstrate that our techniques increase the simulation rate on 7545 Cray XT3 processors from 6.3 to 8.1 simulated years per day. Our results demonstrate how minor code modifications can have significant impact on resulting performance for very large processor counts.

[1]

1. Introduction and Motivation

The Parallel Ocean Program (POP) developed at Los Alamos National Laboratory is an important multi-agency ocean model code used for global ocean modeling. POP uses a finite-difference formulation for the baroclinic component of the timestep and a preconditioned conjugate gradient solver in the barotropic component. Parallelism on distributed-memory computers is supported through the Message Passing Interface (MPI) standard. The resolution of POP is typically limited to a rather coarse 1° resolution for climatological studies to enable the simulation of long time-scale events which require a minimum of five simulated years per wall-clock day. However, it has been demonstrated [15] that a very high-resolution ocean simulation, 0.1° at the equator, improves the accuracy of a large number of climatologically important processes. We therefore concentrate on improving the simulation rate of a 0.1° POP benchmark on very large parallel systems.

In our earlier work [8], we demonstrated that it is possible to significantly increase the performance of POP by rewriting the conjugate gradient solver. We used an alternative data structure to reduce the amount of data that must be loaded from the memory hierarchy to the CPU or passed between neighboring MPI processes. The new data structure eliminates the grid points in the computational mesh that correspond to land points. Our changes only require altering a small number of source files.

In this paper, we concentrate on eliminating the load imbalance across processors that is also caused by the presence of land points within the computational mesh. We use an inverse space-filling curve (SFC)-based partitioning algorithm. Space-filling curve based partitioning is a simple and effective partitioning technique that enables the generation of load-balanced partitions. We describe a new space-filling curve, the *Cinco* curve, which when combined with Hilbert and meandering-Peano curves, provides the flexibility to apply space-filling curve partitioning to a much larger set of problem sizes.

Using several partitioning algorithms and the modified conjugate gradient solver, we provide a comparison of simulation rates for the 0.1° POP benchmark. We discover that our modifications enable a significant increase in simulation rate for the 0.1° POP benchmark on $\mathcal{O}(10,000)$ processor systems. In Section 2, we provide the background and review related work. In Section 3, we describe the Parallel Ocean Program in greater detail, which includes a description of data structures in Section 3.1. In Section 4, we describe space-filling curve partitioning. In Section 5, we provided execution times for the 0.1° POP benchmark on

[1]The work of this author was supported through the National Science Foundation Cooperative Grant NSF01 and the Department of Energy, CCPP Program Grant #DE-FC03-97ER62402.

several very large systems. In Section 6, we provide conclusions and directions for further investigation.

2 Background and Related Work

The Parallel Ocean Program is a global ocean model based on a finite-difference formulation. It uses a structured grid that addresses complex boundary conditions by masking out grid points. Masking out grid points is a common technique used by the HIROMB [19], MICOM [5], and OCCAM [10] ocean models. The POP computational mesh is decomposed into rectangular blocks that are partitioned across processors. One or more blocks may be assigned per processor. POP currently supports two partitioning algorithms: a rake algorithm [16], and a cartesian algorithm. The rake algorithm is only applicable if there is more than one block per processor. The rake algorithm initially starts with a cartesian partitioning and then moves blocks to improve load balance. The existing rake algorithm either does not generate a sufficiently load-balanced partitioning or fails to generate a partitioning for large processor counts. We do not consider the current implementation of the rake algorithm because it is not sufficiently robust. Because cartesian partitioning does not differentiate between the land and ocean points, the cartesian algorithm can generate large load-imbalances. Rantakokko [20] uses variable-sized rectangular blocks to minimize inclusion of land points. POP does not currently support variable-sized blocks, but rather a fixed block size that is set at compile time. Data structures whose size is know at compile time enable the compiler to apply additional loop optimizations that increase single processor performance.

An inverse space-filling curve (SFC)-based algorithm is an effective partitioning technique based on the mapping of a two-dimensional space to a one-dimensional line. The mapping maintains proximity of the partitioned computational mesh. The generation of a load-balanced partition is simplified into the partitioning of a one-dimensional line into equal-length segments. It has been successfully applied to partitioning graphs [3] and in parallel adaptive mesh refinement [4, 9, 17]. It has also been applied with success to static partitioning problems [2, 7]. We do not consider a Metis [12] based partitioning because it was observed in [14] that Metis generates load-imbalanced partitions when the number of blocks per processor is $\mathcal{O}(1)$. We describe a new space-filling curve, the *Cinco* curve, which when combined with Hilbert and meandering-Peano curves [22], provides the flexibility to apply space-filling curve partitioning to a much larger set of problem sizes.

The performance of POP and the implication of data structure choice has been analyzed by several groups. Jones [11] describes the addition of a more flexible data structure that allows efficient execution of POP on both cache and vector processors. Wang [24] describes code modifications to POP that improve performance on a specific machine architecture. Kerbyson [13] developed an analytical model of expected parallel performance. Snavely [23] employs a convolution-based framework that uses hardware performance counts and MPI library traces to predict parallel performance. Analytical and trace-based techniques depend on a description of the current resource requirements of the application. The amount of time to perform a fundamental unit of computation on a single processor or the amount of data that must be passed between MPI processes is required input. These models assume that the computational resource requirements are fixed. We instead look for code changes that can reduce the resource requirements. While we demonstrated in [8] the utility of automated memory analysis to examine the impact of data structure changes on the resource requirements, we do not use an analytical or trace-based performance model to predict parallel performance.

3 Parallel Ocean Program

We next describe the computational characteristics of the POP model. The POP timestep is composed of a baroclinic component that uses finite-difference and a barotropic component. An update of all state variables occurs at the end of the timestep. The baroclinic component consists of large sections of embarissingly parallel calculations that involve the streaming of data from the memory hierarchy to the CPU. The execution rate of the baroclinic component is entirely dependent on the quality of the compiler and the characteristics of the memory hierarchy. The barotropic component uses a preconditioned conjugate gradient solver to update the two-dimensional surface pressure. POP uses the single product conjugate gradient [6] to improve scalability of the solver. While the baroclinic component of the solver demonstrates excellent scaling, the conjugate gradient and update of state variables limits POP scalability. Historically, the performance of POP has been sensitive to message latency and the cost of global reduction operations. We have discovered that three separate modifications to the source code enable significant increases in scalability at large processor counts: an update of the boundary exchange routine, new data structures within the barotropic solver, and an alternative partitioning algorithm. The updated boundary exchange routine amounts to message aggregation for the three-dimensional state variables. This trivial change significantly reduces POP sensitivity to machine latency. We mention a second code modification, which involves the addition of a new data structure within the barotropic solver, to illustrate the impact that reducing resource requirements has on code scalability. A more detailed description of the data structure changes is provided in [8]. However, the fo-

cus of this paper is to describe the impact of a new partitioning algorithm within POP. We first start with a description of the POP data structures in detail.

3.1 Data Structures within POP

POP uses a three-dimensional computational mesh. The horizontal dimensions are decomposed into logically rectangular two-dimensional (2D) blocks [11]. The computational mesh is distributed across multiple processors by placing one or more 2D blocks on each processor. There are km vertical levels associated with each 2D horizontal block. Blocks that do not contain any ocean points or land blocks are eliminated. A 1° grid divided into 2D blocks of 20×24 grid points is illustrated in the top panel of Figure 1. The white area represents land; ocean is indicated by gray. Each 2D block contains $bsx \times bsy$ internal grid points and a halo region of width *nghost*. The halo region is used to store grid points that are located in neighboring 2D blocks. The bottom panel of Figure 1 illustrates the 2D data structure within the source code for the block that contains the Iberian peninsula. Note that the large white structure on the upper right side of the image is the Iberian peninsula, while the white structure in the lower right side is the northern coast of Africa. For this particular block, only about 60% of the grid points represent ocean. Table 1 contains the dimensions of the 1° and 0.1° grids. The variables nx and ny are the global numbers of points in the x and y directions of the grid respectively. Table 1 also contains the block sizes bsx and bsy, the number of blocks $nblocks$, and the percentage of land blocks eliminated. Note that percentage of land block elimination is significantly greater in the 0.1° grid versus the 1° grid.

grid {nx,ny,km}	{bsx,bsy}	nblocks	land blocks	$Nb = 2^n 3^m 5^p$ Nb	n	m	p
1° grid							
{320,384,40}	{40,48}	64	0%	8	3		
	{20,24}	246	4%	16	4		
	{10,12}	904	12%	32	5		
0.1° grid							
{3600,2400,40}	{144,96}	541	13%	25			2
	{120,80}	764	15%	30	1	1	1
	{90,60}	1312	18%	40	3		1
	{72,48}	2009	20%	50	1		2
	{60,40}	2822	22%	60	2	1	1
	{48,32}	4324	23%	75		1	2
	{45,30}	4884	24%	80	4		1
	{36,24}	7545	25%	100	2		2
	{30,20}	10705	26%	120	3	1	1
	{24,16}	16528	27%	150	1	1	2
	{18,12}	28972	28%	200	3		2
	{15,10}	41352	28%	240	4	1	1
	{12,8}	64074	29%	300	2	1	2

Table 1. Description of sub-block configurations for 1° and 0.1° grids.

The primary advantage of the 2D data structure is that it provides regular stride-one access for the matrix-vector multiply, which is composed of a 9-point stencil, within the conjugate gradient solver. The disadvantage of the 2D data structure is that it includes a large number of grid points that represent land. While it is possible to reduce the number of excessive land points by reducing the block size, smaller blocks have a larger percentage of their total size dedicated to halo points.

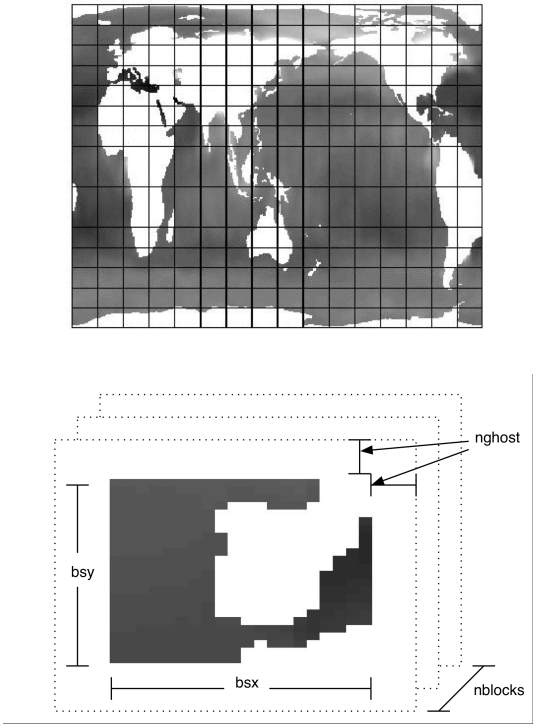

Figure 1. The 1° POP grid, where white corresponds to land points, gray to ocean points, and superimposed lines indicate 20×24 blocks (left). The two-dimensional block over the Iberian peninsula (right).

We implemented a 1D data structure in the barotropic solver which enables the elimination of all land points present in the 2D data structure. The 1D data structure consists of a 1D array of extent *n*. The first *nActive* elements in the 1D array correspond to active ocean points, while the remaining $n - nActive$ points correspond to the off-processor halo needed by the 9-point stencil. In removing the excessive land points, the 1D data structure changes the form of the matrix-vector multiply. Indirect addressing is now necessary to apply the operator matrix that is stored in a compressed sparse-row format.

4 Space-Filling Curve Partitioning

A space-filling curve is defined as a bijective function that maps a line into a multi-dimensional domain. An inverse space-filling curve maps a multi-dimensional domain into a line. There are a variety of space-filling curves [22]. A Hilbert curve can be used to partition a domain of size $Nb \times Nb$, where $Nb = 2^n$ and n is an integer refering to the recursive level. A meandering-Peano (m-Peano) curve and a *Cinco* curve support partitioning, where $Nb = 3^m$ or $Nb = 5^p$ respectively, and where m and p are integers. Figure 2 illustrates level-1 Hilbert, m-Peano, and Cinco curves respectively. It is apparent from Figure 2 that the start and endpoints of all three curves lie on a single axis of the $Nb \times Nb$ domain. This feature allows the curves to be nested, enabling the creation of a Hilbert-m-Peano-Cinco curve that supports partitioning of domains of size $Nb = 2^n 3^m 5^p$. The added flexibility of the Cinco curve allows inverse space-filling curve partitioning to be applied to a much larger set of problem sizes.

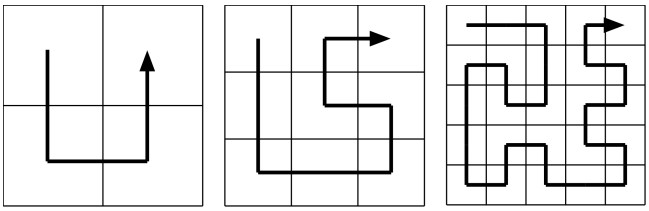

Figure 2. Level-1 Hilbert (left), Meandering Peano (middle), and Cinco (right) curves.

We create an initial space-filling curve using the algorithm described in [7]. Because blocks that contain only land points are eliminated from the ocean simulation, the corresponding segments in the initial space-filling curve must be eliminated. Our line partitioning algorithm subsequently subdivides the resulting space-filling curve into approximately equal length segments. Because our current algorithm does not take into account cuts in the space-filling curve due to land blocks, it has the potential to create a partition such that multiple blocks are not contiguous in the 2D space. For example, two blocks may be separated by a large land mass. Partitions with non-contiguous sets of blocks are sub-optimal because they can increase the communication cost. We examine the impact of non-contiguous sets of blocks on our scaling study in Section 5.

The top panel of Figure 3 illustrates a partition of the 1° grid subdivided into blocks with 20×24 grid points onto 8 processors using a level-4 Hilbert space-filling curve. The bottom panel of Figure 3 corresponds to a single block cartesian partition onto 8 processors.

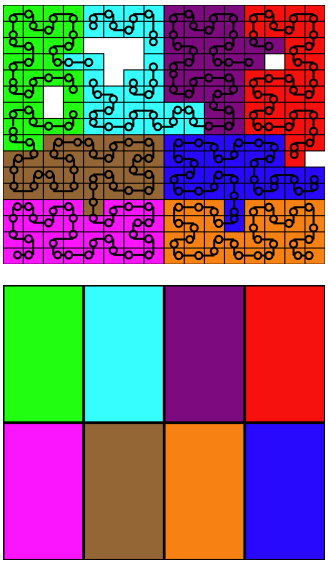

Figure 3. The space-filling curve, which excludes land blocks, provides an ordering of the ocean blocks. A 8 processor partitioning is indicated by the various colors using SFC-based (left) and single block cartesian (right) algorithms.

5 Results

To test the impact of our SFC-based partitioning on the scalability of the 0.1° POP benchmark, we acquired dedicated time on the 40,960 processor IBM Blue Gene Watson (BGW) [1] system at Thomas J. Watson Research and the 10,000 processor Cray RedStorm system at Sandia National Laboratory. Each BGW node consists of a dual-core PPC440 processor that executes at 700 Mhz. The compute nodes are connected by several networks, including high-performance 3D torus for point-to-point messages and a tree network for global reductions and barriers. The Cray RedStorm system is the prototype of the Cray XT3 supercomputer [21]. RedStorm consists of 10,000 nodes connected with a high-performance 3D torus network for message passing. Each node consists of a single 2.0 Ghz AMD Opteron processor. The configuration of each system is summarized in Table 2. We first describe our results on BGW.

We configured the 0.1° POP benchmark to use both single-block cartesian partitioning and several SFC configurations. We tested several block size configurations, which are provided in Table 1. Table 1 provides block sizes in the column $bsx \times bsy$, number of ocean blocks $nblocks$, and the corresponding SFC configuration. The percentage

System		
Name	RedStorm	BGW
Company	Cray	IBM
# of nodes	10,000	20,480
Processor		
CPU	Opteron	PPC440
Mhz	2000	700
Peak Gflops	4.0	2.8
CPU/node	1	2
Memory Hierarchy		
L1 data-cache	64 KB	32 KB
L2 cache	1 MB	2 KB
L3 cache	–	4 MB (shared)
Network		
Network topology	3D torus	3D torus
# of Links/per node	6	6
Bandwidth/link	7600 MB/s	175 MB/s

Table 2. Description of the IBM Blue Gene Watson and Cray RedStorm systems.

of land blocks eliminated is provide in the *land blocks* column. Note that as the block size decreases, land block elimination removes a larger percentage of the total blocks. We configure POP for the SFC tests such that there are approximately equal numbers of blocks on each processor. For example, consider running a job on ∼2400 processors. Table 1 indicates that several block size configurations may work, including 45×30, 36×24, and 30×20. The 45×30 configuration has 4884 blocks, so for a 2442 processor job, two blocks would be placed on each processor. With the 36×24 and 30×20 block size configurations, jobs with 2515 and 2677 processors jobs would place three and four blocks per processor respectively. We find that it is best to look for block size configurations and processor counts that minimize the number of blocks per processor. For the single-block cartesian partitioning, we choose processor counts such that the global domain is evenly divided into block sizes whose x and y dimensions are approximately equal.

We execute the POP 0.1° benchmark, which contains no disk IO, for one wall-clock day or 226 timesteps. The total computational time of the 1-day run, which excludes the startup costs, is used to calculate simulated years per wall-clock day. Figure 4 contains scaling curves for three different configurations on BGW: sub-block with space-filling curve partitioning and the 1D data structure (SFC+1D), single-block cartesian partitioning with the 1D data structure (CART+1D), and single-block cartesian partitioning with 2D data structure (CART+2D). It is clear from Figure 4 that the CART+2D configuration, which corresponds to unmodified POP, has poor scalability above 10,000 processors. The simulation rate increases significantly for CART+1D, which uses the 1D data structure, versus the CART+2D configuration. Interestingly, the increase in simulation rate is entirely due to the rewritten conjugate gradient solver. This result emphasizes the importance of code modifications that reduce the required computational resources. The greatest simulation rate for all processor counts is achieved with the SFC+1D configuration. Both the data structure modifications and SFC partitioning nearly double the simulation rate of the 0.1° POP benchmark on 30K Blue Gene processors from 4.0 to 7.9 years per wall-clock day. On BGW, our modifications achieve the greatest benefit at very large processor counts.

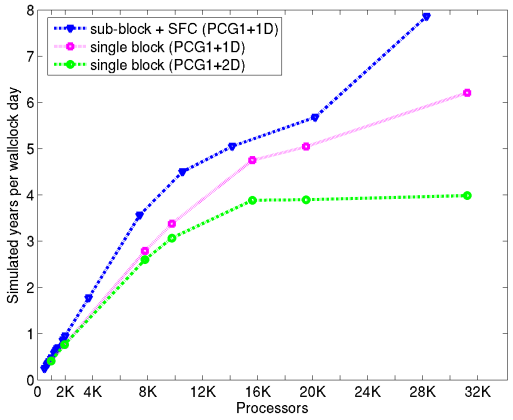

Figure 4. Simulation rate for 0.1° benchmark on BGW with a 2D and 1D data structure based solver and SFC-based partitioning.

A more detailed analysis of our modifications is possible by examining the statistics from several runs on large processor counts on BGW. Table 3 contains statistics from four runs on the Watson system using approximately 29K to 32K processors. The second and third columns in Table 3 contain statistics for two runs that use the single-block cartesian partitioning method, while the fourth and fifth columns contain statistics from runs with SFC partitioning. The average (*AVG*) and maximum (*MAX*) values for the number of active ocean points (*nActive*) and the single processor communication volume (*spcv*) for a single update of a 2D variable is provided for each configuration. Notice that the *MAX*(*spcv*) for the 1D data structure solver is 560 bytes versus 1184 bytes for the 2D data structure solver on 32,000 processors. The reduction in communication volume is a result of only passing the data between MPI processes that is actually necessary. In the baroclinic component of the timestep, a halo width of two is necessary for the advection code. However in the barotropic solver, the 9-point stencil only requires a halo width of one. By only passing the necessary data between MPI processes, the single processor data volume in the solver is reduced by a factor of two. The result

of minimizing resource requirements within the barotropic solver reduces the execution time of the barotropic solver on 32,000 processors from 30.21 to 8.80 seconds.

The computational and communication load imbalance is also provided in Table 3. The load balance (*LB*) for a set $S = \{s_1, s_2, ..., s_n\}$ is calculated as $LB(S) = (MAX\{S\} - AVG\{S\})/AVG\{S\}$. We expect our SFC-based partitioning algorithm to reduce load imbalance. Table 3 indicates that the computational load imbalance *LB*(*nActive*) is, as expected, significantly reduced from 48% to ∼5%, while communication load imbalance *LB*(*spcv*) is similarly reduced from 49% to 8%. Interestingly, even through SFC partitioning increased *AVG*(*spcv*) from 376 to 472 bytes, the *MAX*(*spcv*) dropped, decreasing from 560 to 512 bytes.

Configuration				
nprocs	32,000	32,000	32,038	28,972
$bsx \times bsy$	18×15	18×15	12×8	18×12
blocks/proc	1	1	2	1
partitioning	CART	CART	SFC	SFC
solver	2D	1D	1D	1D
Number of Grid points				
AVG(*nActive*)	182	182	182	201
MAX(*nActive*)	270	270	192	216
LB(*nActive*)	.48	.48	.05	.07
Message Volume in Solver				
AVG(*spcv*)		376	480	472
MAX(*spcv*)	1184	560	576	512
LB(*spcv*)		.49	.20	.08
Time				
3D-update	6.34	6.42	5.78	5.59
baroclinic	22.82	22.89	22.64	19.35
barotropic	30.21	8.80	5.26	5.18
total	59.37	38.11	33.68	30.12
Simulation Rate				
years/day	3.98	6.21	7.03	7.86
GFlops	1023	1593	1803	1967
% peak	1.1%	1.8%	2.0%	2.4%

Table 3. Description of different run configurations on BGW.

Blue Gene has profiling libraries that provide the number and size of each MPI message and access to network hardware performance counters. The network hardware performance counters provides counts of the packets that traverse each node of the torus network. We use the packet counters and message information to approximate the average distance a message travels within the Blue Gene torus network. The number of packets for each of the six networks links for the CART+1D on 32,000 processor configuration and SFC+1D on 28,972 processor configuration is provided in Table 4. Both configurations execute on a $32 \times 32 \times 16$ node torus, which has a network diameter of 20 links. Network diameter is the maximum distance in network links from one node to another. Note that the SFC+1D configuration significantly reduces the average packet count on the negative Y link (Y^-) and positive Y link (Y^+) versus the CART+1D configuration. A smaller reduction in average packet count is seen for the X and Z links. The average total packet count (Σ) for the SFC+1D configuration is 45% less than for the CART+1D configuration. The reduction in average total packet count is achieved despite the fact that the SFC+1D configuration has a larger *AVG*(*spcv*) than the CART+1D configuration.

One possible cause for the reduction in packets for the SFC+1D versus the CART+1D configuration, is that the average distance between neighbors is reduced. We can approximate the average distance between neighbors, by approximating the average number of packets that are injected into the network. Because the number of packets in the network is conserved, it is possible to approximate the mean distance between neighbors. For example if all neighbors were separated by two network links, than the total number of packets in the system should be twice the number injected. Therefore we calculate the average distance between neighbors (Δ_{avg}) by the equation:

$$\Delta_{avg} = \Sigma / I_{avg}$$

where Σ is sum of the packet counts for all links, and I_{avg} is the average number of packets injected into the network.

We approximate I_{avg} based on the MPI message statistics. The Blue Gene torus network uses packets with 224 byte payload, and a rendezvous protocol for messages larger than 1000 bytes. We assume two additional packets are used, to account for the acknowledgment packets in the rendezvous protocol. Our approximation for I_{avg} and Δ_{avg} is provided in Table 4. Surprisingly the average distance between neighbors for the CART+1D configuration is 15.7 hops which is close to the network diameter of 20. It is unclear the reason for the large separation between neighbors and could be related to the routing algorithms. While Blue Gene uses an adaptive routing algorithm for larger messages, the majority of the messages in the POP 0.1° benchmark are smaller than the adaptive routing threshold. The SFC+1D configuration significantly reduces the average distance between neighbors to 9.9 network hops. A reduction in the distance between neighbors reduces the probability of network contention. Unfortunately, due to lack of profiling data, we are unable to approximate the maximum distance between neighbors for each configuration.

While SFC-based partitioning reduces the total number of packets in the system it does not always reduce the maximum number of packets for a particular link. The comprehensive packet statistics provides the opportunity for analysis of the spatial distribution of network congestion within the torus. We use Paraview [18] to visualize hotspots or network links within the torus that have significantly higher packet count. Figure 5 is a visualization of the Y^+ link traf-

fic for both the CART+1D and SFC+1D configuration on a $32 \times 32 \times 16$ node torus. Note the cubes represent the CART+1D configuration while the sphere to the SFC+1D configuration. The color and size of the cubes and spheres correspond to the number of packets on the links. We only render those network nodes whose packet count is within 10% of the maximum packet count in the network. Interestingly, in the case of the Y^+ link, the magnitude and number of hotspots in the network is significantly reduce for SFC+1D configuration. For the CART+1D, a whole section of the network is congested. Similar plots for the other torus links indicates that the congestion in the SFC+1D configuration is similarly isolated to small number of links, while the CART+1D configuration generates large regions of congestion. The impact of the spatial distribution of congestion and whether it could be improved through processor mapping in an avenue for further investigation.

Configuration		
nprocs	32,000	28,972
torus dim.	$32 \times 32 \times 16$	$32 \times 32 \times 16$
torus dia.	20	20
partitioning	CART	SFC
solver	1D	1D
Packets per network node (average)		
X^-	3.3M	2.9M
X^+	3.3M	2.9M
Y^-	2.5M	0.5M
Y^+	2.5M	0.5M
Z^-	0.08M	0.05M
Z^+	0.08M	0.05M
Σ	11.8M	4.1M
I_{avg}	0.73M	0.70M
Average distance in hops		
Δ_{avg}	15.7	9.9

Table 4. Network statistics for two configurations on BGW.

The simulation rate of the 0.1° POP benchmark on the 10,000 processor Cray RedStorm system is also improved through the use of SFC partitioning. A plot of simulation rate as a function of processor count for the 0.1° POP benchmark is provided in Figure 6 using single-block cartesian partitioning with the 2D data structure (CART+2D) and the SFC partitioning with the 1D data structure (SFC+1D) configurations. Unlike on BGW, SFC partitioning provides significant increases in simulation rate for all processor counts. It is interesting to note that the best simulation rate of 8.1 years per wall-clock day occurs on 7545 processors.

The reason for the decrease in simulation rate above 7545 processors is clear if we examine the statistics from the 7545, 8264, and 9658 processor runs provided in Table 5. Table 5 indicates that the communication-intensive components of POP, the 3D-update at the end of the timestep, and the barotropic solver costs increase as processor counts

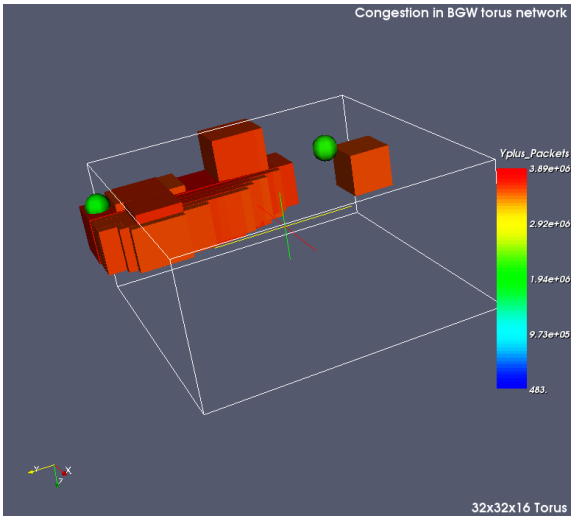

Figure 5. Congestion in the BGW network for a 0.1° POP run on $32 \times 32 \times 16$ node torus. The spheres indicate location and magnitude of packet counts for SFC-based run, while the cubes correspond to CART-based run.

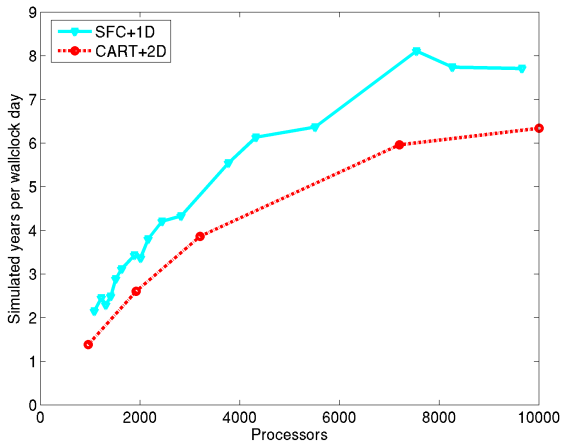

Figure 6. Simulation rate for 0.1° benchmark on RedStorm with a 2D and 1D data structure based solver and SFC-based partitioning.

increase. Increases in the execution time are consistent with increases in *MAX*(*spcv*), which increases from 992 to 1128 bytes. These results clearly demonstrate that the choice of block size impacts the performance advantage that SFC partitioning provides. Our experience and Table 5 confirm that the SFC configuration with the smallest number of blocks minimizes communication and computational costs. SFC configurations with a larger number of blocks increase the possibility of partitions with non-contiguous blocks and higher communication costs. It is possible to provide an estimate on the impact non-contiguous blocks have on performance. Consider the 8264 processor configuration with two blocks of size 24×16. In a partition with non-contiguous blocks, the maximum possible single processor communication volume (*spcv*) would be twice the perimeter of each block. The minimum *spcv* occurs when the two blocks share the longest side of the block. We therefore calculate that for two blocks of size 24×16 the minimum *spcv* is 928 bytes, and the maximum is 1344 bytes. For our current partitioning algorithm, we observed 1088 bytes. An optimal partitioning algorithm could therefore reduce *MAX*(*spcv*) by 15% for the 8264 processor configuration. Similar calculations for the 9658 processor configuration suggests a 20% reduction.

Tables 3 and 5 also allow for comparison of the impact of various architecture features on simulation rate of the 0.1° benchmark. Consider the cost of each component of the POP timestep on 9,658 processors of RedStorm versus 28,972 processors of Blue Gene. The cost of the communication-intensive 3D-update is nearly identical on both systems: 5.68 on RedStorm versus 5.59 seconds on BGW. Because the 3D variables are updated using the 2D boundary exchange, we can accurately compare the cost of the 3D-update to the cost of the 2D updates within the barotropic solver. The cost of the floating-point intensive baroclinic component is 11.78 seconds on RedStorm versus 19.35 seconds on BGW. Interestingly, the execution time of the barotropic component is significantly higher on RedStorm; 13.25 seconds versus 5.18 seconds on BGW. The execution time for the barotropic solver is composed of the time to perform boundary exchanges, floating-point arithmetic, and global reductions. The cost of the 3D-update and the baroclinic component suggest that RedStorm should have a slightly lower cost for the boundary exchange and floating-point arithmetic cost. The difference in execution time for the barotropic component is therefore a direct result of the high-performance dedicated reduction network on BGW. These results suggest that the simulation rate of the 0.1° POP benchmark could be significantly increased on the Cray XT3 through the addition of a high-performance dedicated reduction network. Alternatively, the BGW system could benefit from a microprocessor with improved sustained floating-point performance. The 28972 processor run on BGW only sustains 1977 Gflops or 2.4% of peak performance. Because the floating-point calculations in POP are distributed across a large number of subroutines, it is not feasible to resort to hand optimized assembly code. It should be noted that for our timing runs, the executable did not contain the SIMD instructions that enable the use of the second FPU on the PPC 440 processor. While tests of POP at 1° indicate that the use of SIMD instructions reduces execution time by $\sim$5%, POP at 0.1° generates SEGV when SIMD instructions are used. Compiler support for the second FPU on the PPC 440 processor needs further improvement.

Configuration			
nprocs	7,545	8,264	9,658
$bsx \times bsy$	36×24	24×16	18×12
blocks/proc	1	2	3
partitioning	SFC	SFC	SFC
solver	1D	1D	1D
Number of Grid points			
AVG(*nActive*)	769	678	579
MAX(*nActive*)	864	768	648
LB(*nActive*)	.12	.13	.12
Message Volume in Solver			
AVG(*spcv*)	888	912	920
MAX(*spcv*)	992	1088	1128
LB(*spcv*)	.12	.19	.22
Time			
3D-update	3.81	5.05	5.68
baroclinic	13.14	12.27	11.78
barotropic	12.24	13.27	13.25
total	29.19	30.59	30.71
Simulation Rate			
years/day	8.11	7.74	7.71
GFlops	2080	1985	1977
% Peak	6.9%	6.0%	5.1%

Table 5. Description of several run configurations on the RedStorm system at Sandia.

6 Conclusion

We demonstrate how several code modifications to the Parallel Ocean Program (POP) significantly impacts the scalability of a high-resolution ocean model on very large processor counts. Our modifications nearly double the simulation rate of the 0.1° POP benchmark on 30K Blue Gene processors. Interestingly, these modifications are not dependent on exotic parallel programing techniques but rather on a basic fundamental principle of reducing resource requirements. Our performance gains are a direct result of reducing the amount of data loaded from the memory hierarchy, reducing the amount of data passed between MPI processes, and evenly distributing the computational workload. Despite the significant gains observed, we still see opportuni-

ties for further improvement. We discovered the need for message aggregation in the 3D-update section of the code after our performance runs were made. We therefore believe that the cost of the 3D-update could be cut at least in half on for both the BGW and RedStorm systems. Additionally, the conjugate gradient solver within the barotropic component uses a simple scaled diagonal preconditioner. We believe that it is possible to construct a scalable preconditioner that could cut the cost of the barotropic component in half. It is interesting to note that the importance of an improved scalable preconditioner on BGW is less critical due to its dedicated reduction network. Cutting the execution time of the solver in half would only reduce execution time by 8% on BGW versus 20% on RedStorm. While it may also be possible to further reduce communication volume through improvements to our curve partitioning algorithm, the possible savings of 15-20% of the communication costs on RedStorm is minimal. Interestingly the network hardware performance counts on Blue Gene indicate that the space-filling curve based partitions tend to generate isolated spots of congestion within the torus network while the cartesian based partitioning generates large regions of network congestion. The impact of network congestion and whether it can be reduced using alternate processor mappings are additional avenues for further investigation.

Acknowledgements

We would like to thank Mark Taylor for running our modified code on RedStorm at Sandia National Laboratory, and Benjamin Kadlec for assistance in visualization of the network hardware counters. We would like to also thank Ed Jedlicka of Argonne National Laboratory and Fred Mintzer of IBM Research for providing access to BGW through the 2^{nd} Blue Gene Watson Consortium Days event. Code development would not have been possible without the access to the Blue Gene system at NCAR, which is funded through NSF MRI Grants CNS-0421498, CNS-0420873, and CNS-0420985 and through the IBM Shared University Research (SUR) Program with the University of Colorado.

References

[1] N. R. Adiga and et al. An overview of the Blue Gene/L supercomputer. In *Proceedings of SC2002*, Baltimore, MD, November 2002.

[2] G. Almasi, G. Bhanot, D. Chen, M. Elefheriou, B. Fitch, A. Gara, R. Germain, M. Gupta, M. C. Pitman, A. Rayshubskiy, J. Sexton, F. Suits, P. Vranas, B. Walkup, T. J. C. Ward, Y. Zhestkov, A. Curioni, W. Curioni, C. Archer, J. E. Moreira, R. Loft, H. M. Tufo, T. Voran, and K. Riley. Early experience with scientific applications on the Blue Gene/L supercomputer. In *Proceedings of Euro-Par 2005, Lisbon, Portugal*, Lisbon, Portugal, 2005.

[3] S. Aluru and F. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *4th IEEE International Conference on High Performance Computing*, pages 230–235, 1997.

[4] Jorn Behrens and Jens Zimmermann. Parallelizing an unstructured grid generator with a space-filling curve approach. In *Euro-Par 2000 Parallel Processing*, pages 815–823, 2000.

[5] R. Bleck, S. Dean, M. O'Keeefe, and A. Sawdey. A comparison of data-parallel and message-passing versions of the Miami Isopycnic Coordinate Ocean Model (MICOM). *Parallel Computing*, 21:1695–1720, 1995.

[6] E. F. D'Azevedo, V. L. Eijkhout, and C. H. Romine. Conjugate gradient algorithms with reduced synchronization overhead on distributed memory multiprocessors. Technical Report 56, LAPACK Working Note, August 1993.

[7] J. M. Dennis. Partitioning with space-filling curves on the cubed-sphere. In *Proceedings of Workshop on Massively Parallel Processing at IPDPS'03*, Nice, France, April 2003.

[8] J. M. Dennis and E. R. Jessup. Applying automated memory analysis to improve iterative algorithms. *SIAM Journal on Scientific Computing: Copper Mountain Special Issue on iterative methods*, 2006. Also published as Department of Computer Science, University of Colorado, Technical Report CU-CS-1012-06.

[9] Michael Griebel and Gerhard Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves. *Parallel Computing*, 25(7):827–845, 1999.

[10] C. S. Gwilliam. The OCCAM global ocean model. In *Proceedings of the sixth ECMWF workshop on the use of parallel processors in meterology*, 1994.

[11] P. W. Jones, P. H. Worley, Y. Yoshida, J. B. III White, and J. Levesque. Practical performance portability in the Parallel Ocean Program (POP). *Concurrency Comput. Prac. Exper*, 17:1317–1327, 2005.

[12] George Karypis and Vipin Kumar. Metis - a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of space matrices. web, September 1998. http://www-users.cs.umn.edu/ karypis/metis/(1998).

[13] D. J. Kerbyson and P. W. Jones. A performance model of the parallel ocean program. *International Journal of High Performance Computing Applications*, 19(3):261–276, 2005.

[14] Richard D. Loft, Stephen J. Thomas, and John M. Dennis. Terascale spectralelement dynamical core for atmospheric general circulation models. In *Proceedings of SC2001*, 2001.

[15] M. E. Maltrud and J. L. McClean. An eddy resolving global 1/10 degree POP ocean simulation. *Ocean Modeling*, 8(1-2), 2005.

[16] C. P. Marquet and J. L. Dekeyser. Data-parallel load balancing strategies. *Parallel Computing*, 24:1665–1684, 1998.

[17] Manish Parashar. Enabling distributed, adaptive and interactive applications, 2002. http://www.caip.rutgers.edu/TASSL.

[18] Paraview: Parallel visualization application, 2007. http://www.paraview.org/HTML/Features.html.

[19] J. Rantakokko. A framework for partitioning structured grids with inhomogeneous workload. *Parallel Algorithms and Applications*, 13:135–151, 1998.

[20] J. Rantakokko. An integrated decomposition and partitioning approach for irregular block-structured applications. In *Proceedings of 15th IPDPS Workshop 2000*, Cancun, Mexico, 2000.

[21] The Cray XT3 Supercomputer, 2006. http://www.cray.com/products/xt3/index.html.

[22] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[23] A. Snavely, X. Gao, C. Lee, L. Carrington, N. Wolter, J. Labarta, J. Gimenez, and P. Jones. Performance modeling of HPC applications. In *Parallel Computing (ParCo2003)*, 2003.

[24] P. Wang, D. S. Katz, and Y. Chao. Optimization of a parallel ocean general circulation model. In *Proceedings of SC'97*, November 1997.