Towards Implementation of the Colorado State University Global Cloud Resolving Model on the NSF Blue Waters System

David Randall and Ross Heikes

Department of Atmospheric Science, Colorado State University

Abstract

The Randall team has ported key components of the Colorado State University (CSU) Global Cloud Resolving Model (GCRM) to the Blue Waters system. With the model on Blue Waters there are two levels of parallelism to explore. The first is a coarse grain MPI based communication between computational cores. This parallelism works in conjunction with our global domain decomposition. The team's results show scaling characteristics of the Blue Waters system to 40K cores, and include comparisons with other computer systems. The second level of parallelism is a fine scale parallelism which utilizes the NVIDIA Kepler accelerators. This loop based parallelism directly modifies the numerical operators used within the model. The team has shown that the parallel efficiency of the accelerator strongly depends on the problem size, and has devised modifications to the model to better utilize the accelerators.

Introduction

This report will have three main parts. First we include some background material discussing the model grid, domain decomposition and model equations. Second we include some timings that show the strong scaling characteristics of the MPI based code. And third we show modifications to the code to utilize accelerators.

Grid Background

Here we describe the horizontal grid structure used within the CSU GCRM. Our model is based on an icosahedral grid. We use a horizontal domain decomposition to

achieve coarse grain parallelism using MPI. Figure 1a shows an icosahedron projected

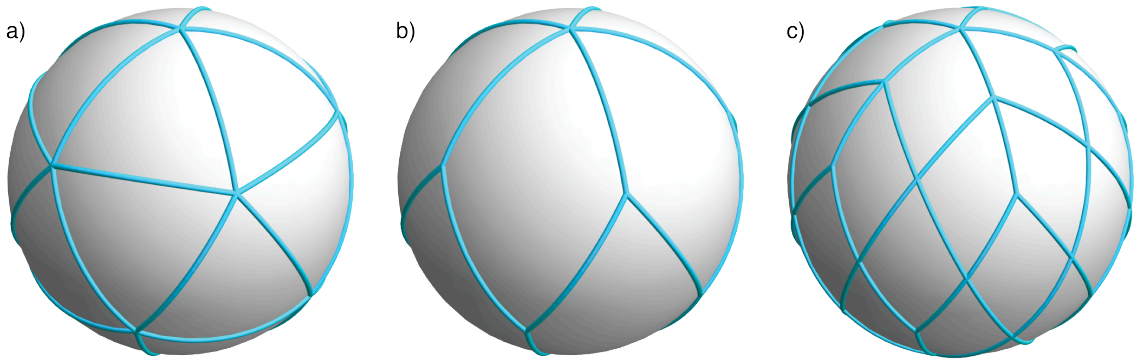

Figure 1. a) Icosahedron projected to unit sphere. 20 spherical triangles. b) Joining pairs of adjacent spherical triangles to form 10 quadrilateral regions.c) Continuing to form 40 regions.

to the sphere. It consists of 20 spherical triangles. Joining adjacent pairs of spherical triangles will form 10 quadrilateral shapes that cover the sphere. This is shown in figure 1b. Each of the 10 quadrilateral regions can be partitioned into four regions as shown in figure 1c. We can continues this process to partition the surface of a sphere.

An recursive algorithm similar to the above domain decomposition algorithm is used to generate the locations of the model control volumes or cells. Cells are assigned to MPI tasks using the above domain decomposition. Figure 2 shows a grid with 642 partitioned into blocks with 64, 16 and 4 cells. Figure 2a corresponds to figure 1b. We call these blocks subdomains. Note that the subdomains are logically rectangular and

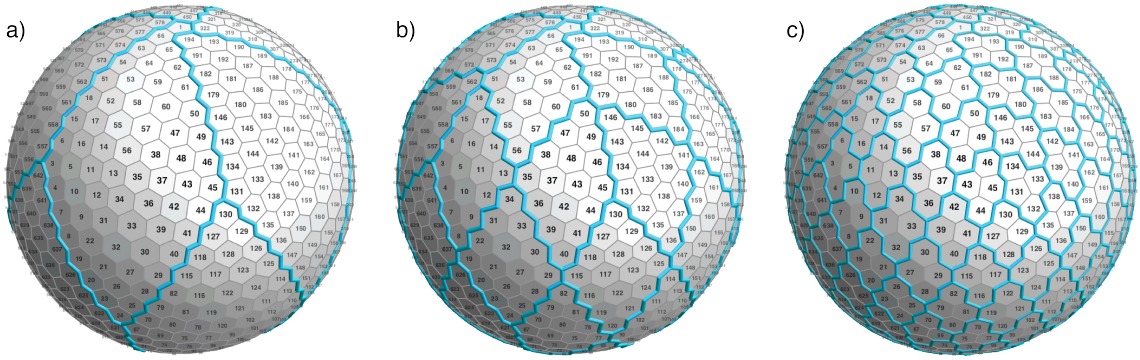

Figure 2. Icosahedral grid with 642 cells partitioned into blocks of a) 64 cells. b) 16 cells. c) 4 cells.

can be stored in conventional 2D arrays. A Morton style numbering of the subdomains and the cells within a subdomain allows physically contiguous subdomains to be also logically close. Numerical finite-difference operators require information from neighboring subdomains to fill ghost cells or halo cells. The information to update boundaries is communicated between blocks using MPI non-blocking sends and receives.

The actual resolution of the model used for numerical simulations has several

resolution (r)	global number of cells	global grid point spacing (km)
9	2,621,442	14.99
10	10,485,762	7.495
11	41,943,042	3.747
12	167,772,162	1.874

Table 1. Target grid resolutions, number of cells and grid spacings.

million cells in each horizontal layer. Table 1 shows the horizontal model resolutions used in this study. The first column show the recursion depth of the algorithm used to generate the grid. Each recursion increases the number of cells by about a factor of four from the previous resolution. The table shows resolutions 9 through 12. These are our target resolutions. This number is used to refer to a particular grid resolution in the following text. The table also shows the global number of cells, and an approximate grid spacing between cell centers. The model is typically run with 32 to 256 layers in the vertical direction.

Model Background

Our model predict vorticity and divergence as prognostic variables. In order to calculate the actual wind for purposes of scalar advection, we need a stream function and a velocity potential. The vorticity ζ and the stream function ψ are related through the elliptic equation $\nabla^2\psi = \zeta$. Similarly, the divergence δ and the velocity potential χ are related through the elliptic equation $\nabla^2\chi = \delta$. The equations for stream function and velocity potential are solved every time step, so an efficient solution is very important. The elliptic equations are solved with 2D multigrid in each model layer. We have determined that the smoothing operator within the multigrid algorithm consumes a significant portion of the total model run time. This will be our first target for improvement. For example, we have run the model to simulate an idealized tropical cyclone with simplified physics. This simulation was run with a global resolution of about 16km on 160 cores. In this case, the smoothing operator on the finest grid accounts for about 11.5% of the total computation. This percentage is approximately constant for varying grid resolution and number of cores. It is a very significant portion of total time considering the numerous other physical processes in the model.

MPI Timings

We have investigated the strong scaling characteristics of the 2D multigrid with a series of numerical experiments. In these experiments the multigrid code was run in

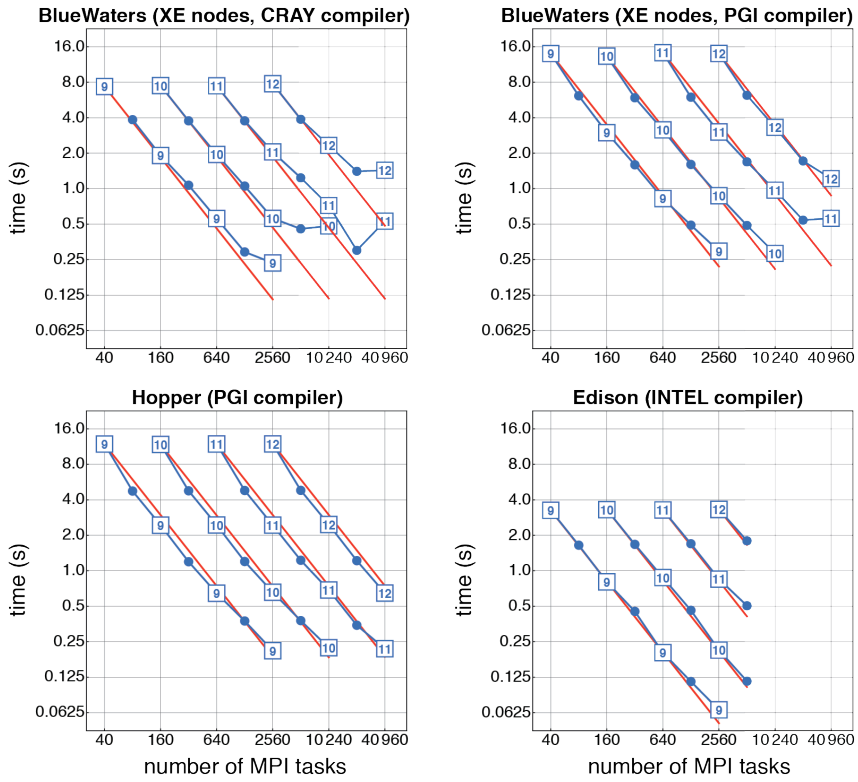

Figure 3. Scaling Characteristics on BlueWaters, Hopper and Edison.

standalone mode with varied resolution and number of MPI tasks. Figure 3 shows the results of these experiments. The plots show the time required to perform 10 multigrid V-cycles with 32 vertical layers with 40 to 40960 MPI tasks. Each blue line shows a particular grid resolution for grids 9, 10, 11 and 12. The red lines show the idealized speed-up. For this comparison we ran the code on Blue Waters, Hopper and Edison. Hopper and Edison are at the National Energy Research Scientific Computing Center (NERSC). Hopper is a CRAY XE6 and Edison is a CRAY XC30. On Blue Waters the CRAY compiler is about twice as fast as the PGI compiler, however the PGI compiler seems to scale better. Hopper shows better scaling characteristics than Blue Waters and scales well to 40K cores. Edison scales well and is faster than Blue Waters and Hopper, however the current configuration is limited to about 10K cores.

We can gain some insight into these scalings by examining how long each core takes to complete a global boundary update. For grid 11 we ran a standalone version of the global communication code for ghost cell updates with 2560, 10240 and 40960 cores. We timed the total time and time to compete portions of the MPI code. This is

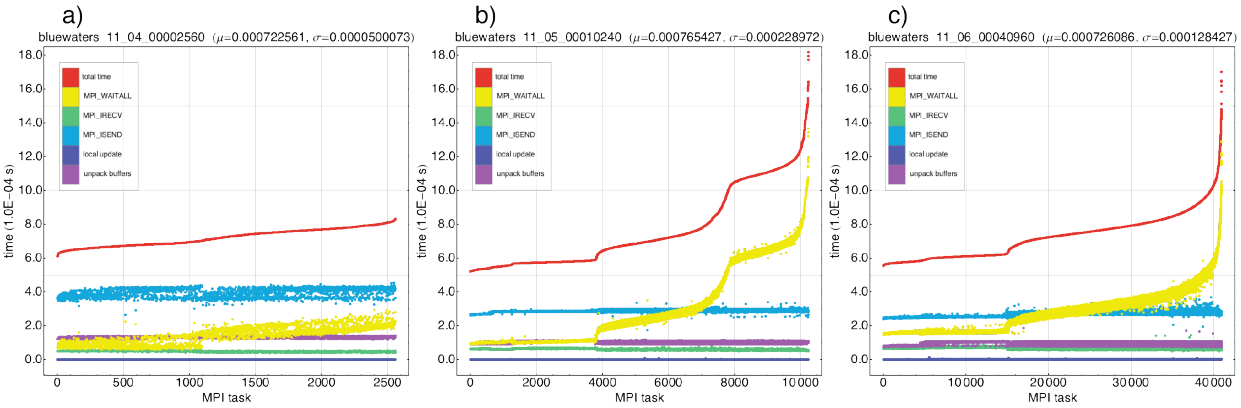

Figure 4. Time per model column for MPI global boundary update of grid 11 with a) 2560, b) 10240 and c) 40960 cores. The data are sorted by total time.

shown in Figure 4. Each figure shows the time per model column. Ideally this should constant and independent of number of cores. The data are sorted according to total time into increasing order as shown by the red line in each plot. The yellow line is the time spend in the MPI_WAITALL routine, and the blue line is the time spent in the MPI_ISEND routine. Other routines are included in the timing but they are insignificant. For 2560 cores the total time is somewhat constant for all cores, and the wait time is small compared the the time to initiate sends. For 10240 cores the discrepancy in times dramatically changes. The fastest times stay fairly consistent, but the longest times dramatically increase. There are a relatively small number of outliers that are more than three times slower than the fastest time. The increase is primarily due to the time spent waiting for messages to complete. Since all the cores must wait for the slowest cores to complete, these cores dominate the overall timings. With increasing number of cores there is greater likelihood that physically close cores are not close within the computer network. It is difficult to control the network distribution of cores within the machine, but better control could improve this discrepancy.

ACC implementation

Our goal is to create a version of the two-dimensional elliptic solver used in the CSU icosahedral grid atmospheric dynamical core that utilizes the GPUs on Blue Waters. The most computationally intensive portion of the multigrid algorithm is the relaxation operator. The relaxation operator is global so it requires MPI communication between subdomain blocks as well as nearest neighbor communication within a subdomain

block. We have performed several numerical experiments using the relaxation operator

```
SUBROUTINE mltgrd2D_rlx (lvl,itermax,im0,jm0,km0,nsdm0,area,wght,beta,alph)

!$acc data copyin (om1,om2,area,wght,beta) create (tmpry,work) copy (alph)

DO iter = 1,itermax ! number of sweeps
```

MPI communication

```
!$acc update device (alph(1:im0-1, 1     ,:,:))
!$acc update device (alph(  im0   ,1:jm0-1,:,:))
!$acc update device (alph(2:im0   ,  jm0  ,:,:))
!$acc update device (alph(   1    ,2:jm0  ,:,:))

!$acc kernels
```

Relaxation sweep

```
!$acc end kernels

!$acc update host (alph(2:im0-2, 2     ,:,:))
!$acc update host (alph(  im0-1,2:jm0-2,:,:))
!$acc update host (alph(3:im0-1,  jm0-1,:,:))
!$acc update host (alph(   2   ,3:jm0-1,:,:))

ENDDO ! iter

!$acc end data

END SUBROUTINE mltgrd2D_rlx
```

Figure 5. Schematic code for relaxation operator.

to better understand its behavior. This block of code is shown schematically in Figure 5.

The ACC **`data`** directive transfers information about the entire subdomain block from the CPU to the accelerator. The outer loop determines the number of relaxation sweeps. The blue box labeled *MPI communication* represented the global message passing between subdomain blocks. After the global MPI communication it is only necessary to communication the edges of subdomain blocks before the next global communication. With the ACC **`update`** directive only the edges of subdomain blocks are transferred between the CPU and the GPU. Data are moving between the CPU and the GPU through the PCI Express bus. The transfer is a relatively slow computational bottleneck, but once the data are on the GPU using them in calculations is very efficient relative to the CPU. By loading (or unloading) the appropriate modules and altering the makefile to include the appropriate compiler directives, the relaxation sweep will be performed on either the CPU or the accelerator.

The efficiency of a relaxation sweep depends on the the size of a subdomain block. To demonstrate this we show numerical experiments comparing results with varying subdomain size. In the experiments the subdomain sizes have a horizontal extent of 32×32 and 64×64, and both cases have 32 layers in the vertical. We would like each core to have a 16×16 or 32×32 block cells in order to achieve our target level of

parallelism Larger blocks will not allow enough cores. If we consider the 32×32×32 block of cells, the ratio of the surface area to the volume is 1/8. In other words, the ratio of the edge cells to all cells is 1/8. This ratio is a measure of parallel efficiency. That is, within a relaxation sweep 8 cells are updated for each cell data transfer between CPU and GPU. Since a relaxation sweep cell update is useful work and the transfer is not useful work, a smaller ratio is more desirable. So, we would expect the 64×64×32 block, with a surface area to the volume of 1/16, to be more efficient.

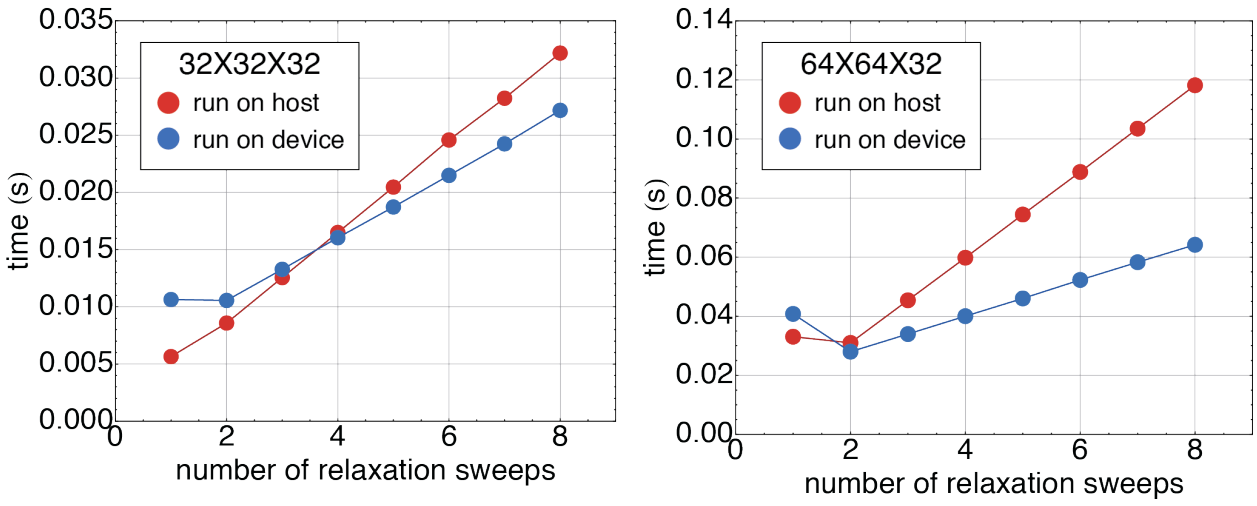

Figure 6. Timing for relaxation sweep run on host and GPU device for varying subdomains size.

Figure 6 shows timings for these two subdomains sizes as a function of number of relaxations sweeps. For the case with 32×32×32 subdomains and one relaxation sweep, we can see the latency resulting from data transfer causes a sweep on the GPU to take twice the time of the CPU. After this initial penalty the timings on both the CPU and the GPU scale more or less linearly with the number of sweeps. With this parallel efficiency the GPU are only marginally faster than the CPU, and with the initial latency, the GPU is only faster than the CPU after four sweeps. Four sweeps is about what is required for sufficient smoothness within the multigrid solver, so in this case it is questionable if there any benefit from using the GPU. This problem will become more acute for coarser grids within a multigrid v-cycle as the subdomain blocks become smaller. Next consider the case with 64×64×32 subdomains. The time per sweep on the CPU is increased by a factor of 3.7 compared to the 32×32×32 subdomains. However, because of the improved parallel efficiency, the time per sweep on the GPU is increased by a factor of only 2.4. And, the time for four sweeps on the GPU is considerably faster than the CPU. So, in this case, the use of the GPU within the multigrid might have some benefit. We are experimenting with a data transpose to allow both a relatively large number cores and subdomains with larger horizontal extent

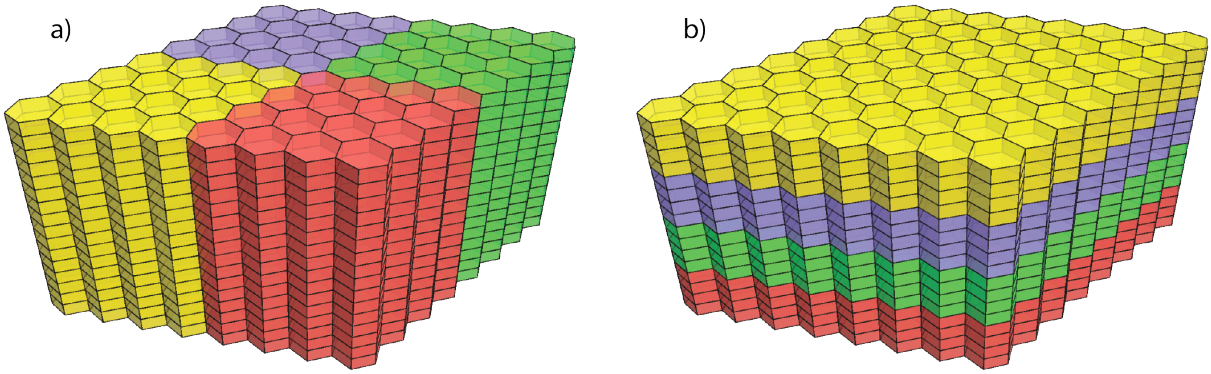

Figure 7. A data transpose. Each color represents the cells of a particular core. a) 4X4X16 subdomains. b) 8X8X4 subdomains.

This is shown in Figure 7. Each application of the transpose doubles the horizontal extent of subdomain. A transpose can be performed during each grid coarsening within the multigrid v-cycle. The coarsening and transpose can be combined into a single operation.

Conclusions

We have shown the strong scaling characteristics of the large scale parallelism on Blue Waters, and made comparisons with other computer systems. We have speculated that reduced scaling efficiency results from discrepancy in time to complete message passing. It would be interesting to quantify the correlation between physical adjacency and message passing time.

We have explored the application of accelerators in the 2D multigrid algorithm, and have shown the that efficiency depends on the size of the problem being solved.