CRC Report No. A-63

SPEED ENHANCEMENTS FOR CAMx PROBING TOOLS

FINAL REPORT

June 7, 2007

COORDINATING RESEARCH COUNCIL, INC.
3650 MANSELL ROAD·SUITE 140·ALPHARETTA, GA 30022

ACKNOWLEDGEMENTS

ENVIRON's CAMx development team acknowledge the support of the Coordinating Research Council, Atmospheric Impacts Committee (CRC-AIC) in developing new science for the publicly available CAMx model.

LEGAL NOTICE

Final Report

CRC PROJECT A-63

Speed Enhancements for CAMx Probing Tools

Prepared for:

CRC Atmospheric Impacts Committee
Coordinating Research Council
3650 Mansell Road, Suite 140
Alpharetta, GA 30022

Prepared by:

Chris Emery
Gary Wilson
Greg Yarwood
ENVIRON International Corporation
101 Rowland Way
Novato, CA 94945

June 7, 2007

TABLE OF CONTENTS

Page

EXECUTIVE SUMMARY

The objective of this project was to continue to revise the Comprehensive Air quality Model with extensions (CAMx) so that multiple computer processors can be used to increase the speed of model simulations that involve Probing Tools. The multi-processor capability implemented in this revision uses the Message Passing Interface (MPI) protocol that allows multiple networked computers to share the load in integrating the model solution. Past support by the Midwest Ozone Group (MOG) and the U.S. Environmental Protection Agency (EPA) allowed us to incorporate multi-processing into the core model. This work sponsored by the CRC extended multi-processing to the source apportionment (SA) and decoupled direct method (DDM) components of the model. The work was arranged according to the following three tasks: (1) initial MPI implementation in the Probing Tools; (2) MPI refinements and optimization; and (3) guidance and documentation. The product of this task is this final report and the delivery of the next generation of the CAMx/MPI beta version. The CAMx User's Guide for version 4.30 (the version upon which this MPI beta is based) is provided with this report. Instructions and guidance for installing, compiling, and using the MPI multi-processing capability in this beta version is provided later in this report.

Under Task 1, initial tests were conducted to ensure that similar Probing Tool results were achieved using one processor vs. multiple processors. These revealed a myriad of problems, ranging from compilation problems, to coding bugs, to significantly different results between applications on one vs. varying numbers of multiple processors. Significant resources were expended to investigate these problems, to develop/consider approaches for refining the code, to implement one to several alternative modifications, and to test the changes. Furthermore, EPA staff and contractors, using their earlier CAMx/MPI beta version, provided very important feedback on technical problems and issues that further allowed us to improve and/or bolster various other aspects of the MPI implementation.

Under Task 2, we developed ideas to streamline/optimize the MPI implementation for speed. While several improvements were incorporated that related to reducing memory images for each type of message passing, limited resources remaining upon completing Task 1 restricted our ability to incorporate all improvements that we identified. Assessments of possible approaches to include the Plume-in-Grid (PiG) algorithm in the MPI implementation have raised significant technical problems; we have been unable to identify a workable approach to include PiG into the MPI structure without adding significant MPI send/receive passes and accounting overhead, thereby defeating any efficiency gains.

Testing MPI speed performance is best gauged for large model applications (i.e., number of grid cells, species, domains) such that time spent on "overhead" tasks such as model setup, I/O, and memory management are minimized relative to the time spent integrating the model forward in time through all the physics and chemistry. In testing non-Probing Tool applications, therefore, we adopted the EPA's photochemical/PM model configuration, which includes a very large two-grid application covering the entire eastern U.S. The model was run for three days, June 21-23, 2001: in single-processor mode, the model took 10.7 hours/day. Distributing the run over 8 processors (4 dual-core machines) reduced the run time to 1.8 hours/day, a speedup factor of 6. Note that we cannot expect the speedup factor to match the number of processors used in an MPI application, since the model overhead processes mentioned above, plus the network overhead to

transfer data fields among the various processors twice each timestep, reduce efficiency. The factor of 6 achieved for 8 processors is considered quite acceptable.

To test the Probing Tools, we configured CAMx to run the DDM option to track 250 sensitivity coefficients in gas-phase photochemical mode only. The model was run for the standard CAMx distribution test case, which includes a 36-km national grid and a 12-km grid over the upper Midwest U.S. (referred to as the MRPO domain). A single day from the test case was run on a network of dual-core Linux PCs and compiled using the Portland Group compiler (v6). In single-processor mode, the model took 7.55 hours to run. Distributing the run over 10 processors (5 dual-core machines), the model took 1.38 hours to run, a speedup factor of ~5.5. Given the smaller grid structures in this test and the larger volume of memory required to run DDM with 250 sensitivities, the overhead costs of passing that memory among the various processors increased relative to the non-Probing Tool run described above. Thus, the DDM run exhibited a smaller speedup factor.

The CAMx/MPI beta version provided to CRC with this report remains incomplete and will thus not be distributed publicly. Besides not incorporating the PiG sub-model, there remains certain numerical errors that we have tracked to the passing of information along the "seams" of the decomposed sub-domains upon which each processor works. This leads to small but noticeable differences in ozone (on the order of parts per trillion) and PM (especially nitrate on the order of tenths of a $\mu g/m^3$) between single and multiple-processor runs of the model. Furthermore, we plan to institute additional optimization strategies that we had identified in this work but were unable to complete within the available resources.

The final CAMx/MPI version for public release will be v5.0. We currently expect to have this version available in late 2007 or early 2008. Chapter 11 of the CAMx User's Guide will be updated at that time to include instructions for compiling, configuring, and running CAMx over multiple processors (information on these topics is provided at the end of this report).

1. INTRODUCTION

The objective of this project was to continue to revise the Comprehensive Air quality Model with extensions (CAMx) so that multiple computer processors can be used to increase the speed of model simulations that involve Probing Tools. The multi-processor capability implemented in this revision uses the Message Passing Interface (MPI) protocol that allows multiple networked computers to share the load in integrating the model solution. Past support by the Midwest Ozone Group (MOG) and the U.S. Environmental Protection Agency (EPA) allowed us to incorporate multi-processing into the core model. This work sponsored by the CRC extended multi-processing to the source apportionment (SA) and decoupled direct method (DDM) Probing Tools.

1.1 BACKGROUND

Currently there are two state-of-the-science one-atmosphere air quality models that are widely used by regulatory agencies and stakeholders to address fine particulate (PM2.5), ozone, and visibility issues: EPA's Models-3 Community Multi-scale Air Quality (CMAQ) modeling system; and ENVIRON's CAMx. Over the years CRC has sponsored several projects to enhance the CAMx model, including the addition of PM algorithms and the implementation of certain Probing Tools. Probing Tools extract information from the model as it runs to assist in model diagnostic, sensitivity, and control strategy evaluation. Aside from such benefits as its Probing Tools, 2-way grid nesting, and ease of use, a major advantage of CAMx over CMAQ has been its computational efficiency, which allows much quicker run times on comparable single-processor systems.

With help from Sandia National Laboratory, EPA implemented a multi-processor capability into CMAQ using the Message Passing Interface (MPI) protocol, which dramatically reduces CMAQ run time by distributing the application load across an array of networked computers such as a cluster environment. This allows CMAQ to run significantly faster than CAMx for single grid applications, although its MPI implementation does not include any of the Plume-in-Grid[1] sub-models. For example, an annual run on the Inter-RPO 36-km U.S. grid takes 3-4 days for CMAQ on a multi-node cluster system, and ~2 weeks for CAMx on a dual-CPU Linux PC. Furthermore, the CAMx Probing Tools add significant resource demands, and this limits their usefulness for annual modeling due to extensive run times on single computers. Thus, the implementation of an MPI multi-processor capability in CAMx is critically important to the continued use of the model, especially when employing the Probing Tools for longer-term simulations.

Initial seed funding for CAMx/MPI was provided in 2005 by the MOG, a stakeholder group that consists of utilities, other industries, and local agencies. This funding allowed us to rewrite core portions of the CAMx code in Fortran 90 with dynamic memory allocation for key variable field arrays, and to restructure various segments of the model. Both were major but necessary steps to prepare CAMx for MPI. The product of this effort was a very simple initial MPI implementation for the core model, but it did not treat nested grids. The EPA then funded work that expanded

[1] There are two versions of CMAQ Plume-in-Grid sub-models; the PinG, and the Advanced Plume Treatment (APT). Each are available in different versions of CMAQ.

and improved the initial MPI implementation, including conversion of all remaining arrays to dynamic memory allocation, the addition of nested grids, and improvement of the domain decomposition approach (i.e., identifying those portions of the entire multi-grid domain that are sent to each processor). However, this funding was insufficient to extend MPI to the CAMx Probing Tools, or to optimize the MPI implementation.

1.2 APPROACH

CRC Project A-63 built off of the work funded by MOG and EPA, specifically to extend MPI to the Probing Tools and to the PiG submodel, and to more fully streamline and optimize the MPI multi-processing capability. The work was arranged according to the following three tasks:

Task 1: Initial MPI implementation in the Probing Tools
MPI was extended to the Probing Tools following the approach used for the core portion of CAMx. Initial tests were conducted to ensure that similar Probing Tool results were achieved using one processor vs. multi-processors. Tests were also undertaken to report the CAMx speedup achieved using various numbers of processors. An approach was developed to include PiG, and to streamline/optimize the MPI implementation for speed and accuracy.

Task 2: MPI Refinements and Optimization
The initial MPI implementation was enhanced according to the results of Task 1. The modified code was tested for a regional simulation using the DDM Probing Tool, and the reduction in simulation time for both core model and Probing Tools was determined for several multi-processor configurations relative to runs employing a single processor.

Task 3: Guidance and Documentation
The product of this task is this final report and the delivery of the next generation of the CAMx/MPI beta version. The CAMx User's Guide for version 4.30 (the version upon which this MPI beta is based) is provided with this report. Instructions and guidance for installing, compiling, and using the MPI multi-processing capability in this beta version is provided later in this report.

Section 2 of this report documents the work accomplished in the tasks listed above, including technical obstacles that we were able to overcome and those that will require additional consideration before they can be implemented into CAMx/MPI. Section 3 provides instructions on installing, compiling, and using this latest beta version of CAMx/MPI. Information in Section 3 supersedes the installation and compilation instructions provided in Chapter 11 of the CAMx v4.30 User's Guide, which accompanies this project report.

2. EXPANSION OF MPI IN CAMx

2.1 INITIAL MPI IMPLEMENTATION IN THE PROBING TOOLS

At the conclusion of our MPI work for the EPA in September 2006, an initial version of CAMx/MPI was delivered to EPA for further testing on their computer system. Tests on our specific cluster platform showed that the core model was sped up by nearly a factor of 3 using 8 compute nodes. For the CRC A-63 project, MPI was extended to the Probing Tools (OSAT/PSAT and DDM), following the approach used for the core portion of CAMx. Initial tests were conducted to ensure that similar Probing Tool results were achieved using one processor vs. multiple processors.

Initial testing of the Probing Tools revealed a myriad of problems, ranging from compilation problems, to coding bugs, to significantly different results between applications on one vs. varying numbers of multiple processors. Significant resources were expended to investigate these problems, to develop/consider approaches for refining the code, to implement one to several alternative modifications, and to test the changes. Indeed, many problems that were encountered under this phase of the project were tracked to the MPI implementation in the core model itself. In particular, we uncovered several issues associated with details in how the domain decomposition methodology impacted the manner in which gridded data were passed among the compute nodes of the network and rebuilt into cohesive concentration fields.

Furthermore, as EPA progressed with their independent testing of the CAMx/MPI beta version concurrently with the first phase of this project, EPA staff and contractors provided very important feedback on technical problems and issues that further allowed us to improve and/or bolster various other aspects of the MPI implementation. One very important finding in our correspondence with EPA was that CAMx/MPI did not port easily to other architectures, operating systems, or Fortran compiler versions and brands that differed from our development system.[2] Additional time was spent working out the details of code bugs revealed by the compiler differences and to expand the model's utility across a broader range of platforms and compilers. Thus, this project afforded a very valuable mechanism to further test and improve the robustness of MPI for the core model as well as the source apportionment and DDM Probing Tools. Ultimately, with each improvement we were able to move closer to an acceptable level of performance in terms of meeting our expected concentration threshold differences (e.g., ozone differences between single and multiple nodes within 0.1%). In summary, this CAMx/MPI beta version being delivered to CRC is a vast improvement in efficiency and code robustness over the version delivered to EPA in September 2006.

[2] Our developmental system included a specific chipset (AMD 32-bit), Linux kernal (2.4), and the Portland Group F90 compiler (v6). With EPA's help, enhanced testing of CAMx/MPI was performed using an earlier version of Portland F90 (v5) and the Intel F90 compiler on Linux 32 and 64-bit machines (using 32-bit compilation).

2.2 MPI REFINEMENTS AND OPTIMIZATION

While incrementally improving CAMx/MPI performance for accuracy under the first phase, we also developed ideas to streamline/optimize the MPI implementation for speed. Several approaches were considered to address the following three areas:

1. Reducing the number of times that data fields need to be passed between the various compute nodes to provide updates for domain overlaps, fine grid boundary conditions, feedback to parent grids, and I/O;
2. Modifying the order in which the models' physical and chemical steps are performed in the operator splitting approach to reduce and/or streamline the data passing procedures among the compute nodes;
3. Improving memory and argument passing structures to reduce data packet sizes.

Several improvements related to point (3) were found to be necessary under Task 1 to alleviate memory faults for very large applications being conducted by EPA (i.e., large nested grid applications requiring significant memory). Points (1) and (2) above are intimately related, since the number of passes needed among the compute nodes depends strongly on when certain processes are performed and when they would need certain data from other processors. Given the current structure of the operator splitting approach in CAMx v4.31 as the basis of this CAMx/MPI beta version, the minimum number of send/receive pass pairings are two (one at the beginning of a time step, one at the end). However, we were able to conceive of an alternative operator order that would reduce the number of passes to one per timestep. However, the operator order is also sensitive to when nested grid boundary conditions need to be updated and when nests should feed back information to the parent grids in the 2-way nested structure. We realized that the single-pass send/receive pairing approach would complicate the grid feedback algorithm, and would require significant time to resolve and to recode the model. Given the limited resources remaining upon completing Task 1 of this project, we decided to maintain the current configuration for this beta version of CAMx/MPI.

Assessments of possible approaches to include the PiG algorithm in the MPI implementation have raised significant technical problems. The key reason for this is that the PiG algorithm is a separately functioning model that must remain in lock-step with the grid model and share and exchange large quantities of chemical and environmental information in a very dynamic way at each time step and among all grids. Since in the MPI approach each computer node receives a portion of a particular grid and works on the solution in near isolation (i.e., limiting the number of times information is passed among the computer nodes), inclusion of PiG into the MPI structure would dramatically increase the overhead associated with information transfer among compute nodes by requiring many more data passes among the grids. Additionally, PiG puffs are free to roam throughout the grids, and therefore across portions of grids apportioned to the various compute nodes, which complicates the approach to split the puff population among the nodes (one potential solution is to assign a PiG-only compute node that works on all puffs exclusively). In summary, we have been unable to identify a workable approach to include PiG into the MPI structure without adding significant MPI send/receive passes and accounting overhead, thereby defeating any efficiency gains.

2.2.1 Results of MPI Performance Gains

Testing MPI speed performance is best gauged for large model applications (i.e., number of grid cells, species, domains) such that time spent on "overhead" tasks such as model setup, I/O, and memory management are minimized relative to the time spent integrating the model forward in time through all the physics and chemistry. In testing non-Probing Tool applications, therefore, we adopted the EPA's photochemical/PM model configuration, which includes a very large two-grid application covering the entire eastern U.S.:

Grid 1, 36-km resolution: 148 x 112 x 14 grid cells (entire U.S.)
Grid 2, 12-km resolution: 281 x 242 x 14 grid cells (entire eastern U.S.)
Chemistry mechanism 4 (52 gas and PM species)

CAMx/MPI was compiled and run on a multi-node Linux cluster system using the Portland Group compiler (both v5 and v6). The cluster was dedicated to the CAMx/MPI application, so no other programs were running. The model was run for three days, June 21-23, 2001: in single-processor mode, the model took 10.7 hours/day. Distributing the run over 8 processors (4 dual-core machines) reduced the run time to 1.8 hours/day, a speedup factor of 6. Note that we cannot expect the speedup factor to match the number of processors used in an MPI application, since the model overhead processes mentioned above, plus the network overhead to transfer data fields among the various processors twice each timestep, reduce efficiency. The factor of 6 achieved for 8 processors is considered quite acceptable.

To test the Probing Tools, we configured CAMx to run the DDM option to track 250 sensitivity coefficients in gas-phase photochemical mode only. The model was run for the standard CAMx distribution test case, which includes a 36-km national grid and a 12-km grid over the upper Midwest U.S. (referred to as the MRPO domain):

Grid 1, 36-km resolution: 97 x 90 x 14 grid cells (entire U.S.)
Grid 2, 12-km resolution: 119 x 134 x 14 grid cells (upper Midwest)
Chemistry mechanism 3 (25 gas and 0 PM species)

A single day from the test case was run on a network of dual-core Linux PCs and compiled using the Portland Group compiler (v6). In single-processor mode, the model took 7.55 hours to run. Distributing the run over 10 processors (5 dual-core machines), the model took 1.38 hours to run, a speedup factor of ~5.5. Given the smaller grid structures in this test and the larger volume of memory required to run DDM with 250 sensitivities, the overhead costs of passing that memory among the various processors increased relative to the non-Probing Tool run described above. Thus, the DDM run exhibited a smaller speedup factor.

3. INSTALLING AND USING CAMx/MPI

ENVIRON is providing the source code of the latest CAMx version 4.31 MPI (beta) developed under this project to the CRC Atmospheric Impacts Committee with this final report. In this section we provide instructions to install, compile, and run CAMx with the new MPI capability. Note that these instructions are specifically limited to this specific version of CAMx, and may be replaced with revised or alternate instructions upon the release of additional beta versions or the final public release version. Information in this section supersedes the installation and compilation instructions provided in Chapter 11 of the CAMx v4.30 User's Guide, which accompanies this project report.

3.1 INSTALLING CAMx/MPI

The tar file delivered with this memorandum is compressed. Copy the tar file to the directory to receive the CAMx source code. Issue the following command:

```
tar –xvzf src.v4.31_MPI_Beta.Mar12_2007.tgz
```

All subdirectories and source code files will be exploded into a specific directory structure within the current parent directory.

Currently, the `Makefile` provided with the CAMx source code only supports building an MPI version of CAMx using the Portland Group PGF90 compiler or the Intel ifort compiler.

3.2 PREPARING CAMx/MPI FOR COMPILATION

In order to facilitate the implementation of MPI in CAMx, the source code was significantly restructured to incorporate dynamic memory allocation. This required a migration from the primarily Fortran 77 constructs originally implemented in CAMx, to Fortran 90 constructs. All of the global data structures are now dynamically allocated during the model setup. This means there is no longer any reason to customize the CAMx parameters file (`camx.prm`) for a particular application. The data necessary to allocate array memory space for a given grid configuration are read from the CAMx control file.

We discovered in early performance comparisons that the use of the Portland Group PGF90 Compiler results in a significant speed/performance dis-benefit when local data array structures within subroutines are declared as allocatable arrays (this is not the case for global and argument arrays, nor is this a problem with the Intel compiler). There is considerable overhead in allocating and deallocating these local arrays each time the subroutine is called. For this reason, this version of the CAMx model continues to utilize some basic parameters to statically allocate local arrays. All of these parameters are defined in the `camx.prm` file. We have provided a version of this "include" file with the parameters set to default values that we feel are sufficiently large to accommodate most applications (see the table below for a description of parameters and their default values). However, you may want to customize these values in one of two ways:

1) Increase a value for a large application; or
2) Set the parameters to exactly match your application, thus preventing wasted memory.

Parameter Name	Description	Default Value
MXCELLS	Number of cells in X/Y direction for any grid	281
MXLAYER	Number of layers	20
MXSPEC	Number of species (could be number of radicals, number of input species, or number of model species)	60
MXREACT	Number of reactions (depends on the mechanism; see the user's guide for the value for each mechanism)	217
MXGRID	Number of grids	10
MXPTSRC	Number of point sources	120000

3.2.1 Compiling CAMx/MPI

For the standard public release version of CAMx, each compilation of the model results in an executable file labeled for a specific application that mirrors the `camx.prm` file used to define the static arrays. Since the `camx.prm` file for this version of CAMx is no longer application-specific, we have modified the `Makefile` to produce an executable with the generic name of `CAMx.v4.31.MPI`. However, the mechanism for specifying the name of a specific application is still in place if you find it necessary to generate a version of CAMx with a particular name and configuration. The steps needed to compile CAMx are as follows:

1. Modify the `Makefile` and set the variable `MPI_INST` (which can be found at line 25) to the path of the installation of the `mpich` package. This is necessary so that the compiler can find the needed header files and libraries for MPI.

2. If the compiled model will be run on a 64-bit operating system, you will probably need to remove the "`-Bstatic`" option from the compiler flags of the "pg_linux" section of the `Makefile`. We have found that an executable built with MPI on a 32-bit compiler will fail to run on a 64-bit OS if the `-Bstatic` option is included. This option will build the compiler libraries into the executable, rather than use the default mode of relying on the dynamic libraries installed on the operating system. The `-Bstatic` option generates a portable executable program. The inconvenience is that excluding the `-Bstatic` option will create the necessity of having the Portland Group dynamic libraries installed on each computer that will serve as a master or compute node in the model application. To remove the static option just delete the `-Bstatic` from line 126:

   ```
   make model FC="pgf90" FLGS="-Bstatic -O2 …
   ```

3. Change directory to the `src.v4.31.MPI/MPI/util` directory. Enter the command "`make`". This will build a library containing some support programs written in C that are used in the message passing implementation. These are generic routines and do not need to be recompiled after making changes to the CAMx FORTRAN code. The library needs to be built only one time before the first compilation of the model.

4. Change directory back to the `src.v4.31.MPI` directory. Enter the command "`make pg_linux`" or "`make i_linux`". The executable will be automatically named: `CAMx.v4.31.MPI.pg_linux` or `CAMx.v4.31.MPI.i_linux`, depending on which compiler was used.

3.3 RUNNING CAMx/MPI

The MPI version of CAMx was designed using a "master/slave" parallel processing approach. The CPU on which the program is launched (process ID 0 under MPI) serves as the master node and will not make any actual model integration computations for any part of the modeling domain. This process will perform all of the model setup, the vast majority of I/O, and manage the communication between the compute nodes. Since the master node handles the important I/O, it is the only CPU which needs access to the disk volume containing the input files and the location of the output directory. This approach allows for minimal amount of network traffic to the nodes on the cluster by eliminating the need for the compute nodes to manage NFS mounts. The master node may need access to the LAN for data access, but the compute nodes only need access to the internal cluster network. However, the compute nodes will need access to a copy of the executable program. This can be accomplished in a number of ways: (1) have an NFS mount on the master node accessible to the internal cluster network and launch the model from that location; or (2) port a copy of the executable program, using `rcp` or `scp`, to the user's home directory on each compute node and launch the model from the user's home directory on the master node.

During each model time step, when computations are performed by the compute nodes, some information is written to the diagnostic (`*.diag`) and message output (`*.out`) files. Rather than just eliminate this information altogether, we decided to create node-specific versions of each of these two files and have each compute node write the information to its own version. However, in order to prevent the need to have the output directory available to the compute node across the network, we have written the model so that the node-specific files are created in the current working directory. That means that if the model is launched from an NFS mounted directory, the node specific files for each compute node will all be created in that location. If the model is launched from a user's home directory on the compute nodes, you will have to log in to the specific compute node to view the files.

The MPI version of CAMx uses the same job script as the standard version, with a few modifications to support multi-processing. Here, we explain the approach we have taken. We provide this as guidance; it is not necessary to follow these steps to run CAMx/MPI. Others may have a different way of handling the MPI portion of the execution.

1. Put the CAMx input files somewhere on the network that is accessible by the computer that will serve as the master node. Also, identify a location where the output will be written.

2. Create a directory on the local disk of the master node that will serve as the launching point for the CAMx execution. Copy the standard CAMx job script, and any scripting support programs (such as j2g) to this location.

3. Modify the job script so that the variable identifying the executable program points to the local directory.

4. Modify the job script so that the pathnames for input and output files are correct.

5. Add a section, similar to the one shown below, to the top of the job script.

```
cat << ieof > nodes
10.1.4.2
10.1.4.3
10.1.4.4
10.1.4.5
10.1.4.6
10.1.4.2
10.1.4.3
10.1.4.4
ieof
set numprocs = `wc -l nodes | awk '{print $1+1}'`
```

   This will create a file containing the IP addresses of the computers that will get a computational slice of the domain on which to work. NOTE: THE MASTER NODE SHOULD NOT BE LISTED and will not get a computational slice. The computer on which the program is launched is included in the MPI simulation by default and is given a process ID of 0. Notice that some computers are listed twice. This example is utilizing a cluster containing dual processor computers. The computer must be listed once for each CPU that is to be used in the simulation. Notice that the `numprocs` variable is assigned to a value equal to one more than the number of computers listed. This is because the master node is included in the count of processors. This example will utilize 9 processors: one to serve as the master node and 8 that will receive a computational slice. If the names of the computers are known to the master node, hostnames can be used in place of IP addresses.

6. Change the line that launches the executable program, at the bottom of the script, to:

```
mpirun -v -machinefile nodes -np $numprocs $EXEC
```

   This will use the `mpirun` utility to launch the CAMx model using the file called "nodes" to identify the computers to be included. Of course, this assumes that the `mpirun` utility is in the current user's path. In fact, the `mpich` package must be installed on each compute node and the `mpirun` utility must be available in the user's path on each system. A complete path to `mpirun`, such as `/usr/local/bin/mpirun`, could be used instead.