Final Report

CAMx MULTIPROCESSING CAPABILITY FOR COMPUTER CLUSTERS USING THE MESSAGE PASSING INTERFACE (MPI) PROTOCOL

Work Order No. 582-07-84005-FY08-06
Project Number: 2008-01

Prepared for:

Texas Commission on Environmental Quality
12100 Park 35 Circle
Austin, TX 78753

Prepared by:

Chris Emery
Gary Wilson
Greg Yarwood
ENVIRON International Corporation
773 San Marin Dr., Suite 2115
Novato, CA 94998

August 31, 2008

TABLE OF CONTENTS

Page

1. INTRODUCTION

1.1 OBJECTIVE AND BACKGROUND

The purpose of this Work Order was to provide various modeling support and development activities to the TCEQ. ENVIRON provided support through the following tasks:

- Completed development of a CAMx multiprocessing capability for computer clusters using MPI, installed and tested on the TCEQ's computer system;
- Added the SPLTEM module and PM processing to EPS3;
- Updated the GRDEM module to EPS3;
- Provided CAMx training for new users.

This report documents the work carried out under the first task (CAMx/MPI), while a separate report addresses the EPS3 updates. Introductory training of TCEQ staff on CAMx (non-MPI) was conducted and completed in mid-August, 2008.

1.1.1 Objective

TCEQ is developing new ozone modeling episodes for the Houston ozone nonattainment area using the Comprehensive Air quality Model with extensions (CAMx). In support of this work, TCEQ directed ENVIRON to provide a CAMx multiprocessing capability that employs the Message Passing Interface (MPI) protocol. Over the past three years, ENVIRON had successfully incorporated MPI into most components of a CAMx/MPI beta version except for the Plume-in-Grid (PiG) module. Hence, the objective of this task was to complete the CAMx/MPI beta version to include PiG, unify the beta version with the current public version of CAMx (v4.51), install the system at TCEQ, and test the MPI capabilities on the TCEQ modeling computer cluster. Installation included both the CAMx model and the separate, but required, MPICH2 library needed to run CAMx/MPI. Testing included both computational performance (speed) and computational accuracy. ENVIRON staff traveled to the TCEQ's offices in Austin to help TCEQ staff install and test the MPI version of CAMx.

Multiprocessing (or "parallelizing") refers to distributing a model application to multiple CPUs that share the computational load in integrating the model solution. Previous versions of CAMx allow for parallelization across multiple CPUs on a single server using the Open Multi-Processing (OpenMP or OMP) protocol for shared-memory multiprocessing. OMP distributes calculations for single grid cells (chemistry) or rows/columns of cells (advection/diffusion) to each CPU within the server. PC platforms are available with multiple (2-8) CPUs (or cores) that access common memory; we find that OMP works well for this number of cores. A computer cluster (i.e., multiple network-connected servers that work together virtually as a single computer) can provide many more CPUs but requires a different multi-processing management system called MPI. With MPI, the CAMx modeling grid is divided into sub-regions that are processed on separate servers within a cluster and integrated by passing large data messages among the servers. MPICH2 is an open source implementation of MPI (version 2) widely used in the atmospheric modeling community. CAMx with MPICH2 allows for improved turn-around by reducing modeling run time from the current 2 hours per episode day for Houston;

CAMx speed is most improved when a combination of both OMP and MPI parallelization is employed for a given model application.

1.1.2 Background

Incorporation of MPI into CAMx began in 2005 with initial seed funding provided by a Midwest stakeholder group, which allowed for the restructuring of CAMx code for an initial basic MPI implementation. This included re-writing CAMx in Fortran 90 with dynamic memory allocation necessary to support MPI. The CAMx/MPI development was initiated from the latest version of the code at the time, v4.31. The U.S. EPA then funded work that expanded and improved the initial MPI implementation, including 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). An automotive research group also funded work to extend MPI to the CAMx Probing Tools.

In 2006/07, CAMx/MPI was streamlined and optimized, and a conceptual methodology to include the PiG algorithm was developed (although PiG was not incorporated into MPI). The modified code was tested for a regional simulation with and without Probing Tools, and the reduction in simulation time for both applications was determined for several multi-processor configurations relative to runs employing a single processor. A beta version of CAMx/MPI was distributed by ENVIRON in Spring 2008 to allow the user community to install and test this code on their systems and to report back on problems, accuracy, and speed improvements.

The current work effort funded by TCEQ completes the implementation of CAMx/MPI by incorporating PiG into the parallelization constructs, and unifying the entire model code to the current public release version, v4.51. The CAMx User’s Guide for version 4.50 (the version upon which this CAMx/MPI is now based) should be referenced for basic information on CAMx science algorithms, input/output requirements, and Probing Tools. Instructions and guidance for installing, compiling, and using the MPI multi-processing capability is provided later in this report.

2. FINAL CAMx/MPI DEVELOPMENT

ENVIRON implemented MPI into the PiG module and performed extensive performance and accuracy testing on several of our Linux cluster environments. The fully implemented MPI capability was transitioned to the latest public-release version of CAMx (v4.51). Additional extensive performance and accuracy testing was conducted with the updated model on our cluster environment, as well as on TCEQ's Linux cluster.

Key efforts in this task centered on implementing CAMx/MPI on the TCEQ cluster system and providing hands-on training and subsequent remote support to TCEQ staff as needed throughout the project. ENVIRON staff visited TCEQ to install CAMx/MPI and test the system to gauge speed improvements and to assess accuracy relative to non-MPI applications. During that visit, we also provided an overview lecture to interested TCEQ staff about the system's capabilities and speed improvements. Support services included guiding TCEQ staff in diagnosing the cause of problems and interactively developing/implementing appropriate solutions. Support was conducted via phone and e-mail.

2.1 PiG IMPLEMENTATION APPROACH

The implementation of PiG into CAMx/MPI began with the development of a conceptual approach. The plan first reviewed how PiG works within the current (non-MPI) CAMx code, then identified several alternative approaches for the implementation, and finally concluded with several strategies on how to address and improve processor load balancing issues that might arise in a simulation with a heavy PiG integration burden. Over the past year, this plan has evolved as different technical hurdles were encountered and solved during the actual implementation steps, resulting in a final plan as listed below.

2.2.1 PiG Implementation Plan

Below is a summary review of PiG routines and the process by which PiG is integrated in time within the standard (non-MPI) version of CAMx v4.51:

- PIGINIT and PIGWALK initialize puffs and walk new/existing puffs, respectively. They are called sequentially at the top of EMISTRNS, but only for the finest grid as that is the limiting factor to ensure puffs emit/walk consistent with grid mass. All PiG sources emit, and all new/existing puffs walk, over the entire domain (all grids) for this fine grid time step.

- PIGDRIVE performs chemistry, dry/wet deposition, puff growth, and any mass dumping to the grid; puffs are stationary during this process. This routine performs chemistry specific to the GREASD and IRON options of PiG, and is called by PIGEVOL (an intermediate routine that transfers common block vectors to local arrays), which is further called by CAMx and NESTING after EMISTRNS and before CHEMDRIV. The PiG driver routine works only on puffs in the current grid over the full grid time step.

- Puff chemistry is performed twice, background (grid only) and puff+background. Then the updated total puff concentrations are subtracted from the updated background concentration to yield incremental puff mass. Currently these two steps (and wet deposition) occur within a loop over puffs.

- PIGSAMPL performs PiG concentration sampling if any passive "sampling grids" are established by the user upon model startup. It is called by CAMx and FGAVRG, right after calls to AVERAGE.

The final approach to implement PiG into MPI is summarized below:

- For CAMx/MPI v4.51 delivered to TCEQ, we have implemented a complete MPI methodology for PiG that produces accurate results. Still, the model could encounter load balancing problems in cases where CAMx is run with a high PiG processing burden in a particular area of the grid. Ideas for PiG load balancing improvements are listed below. If indeed load balancing for PiG presents a problem in this version of CAMx/MPI, the load balancing recommendations will be investigated further in follow-on versions.

- The basic idea is to have each compute node work on the puffs occupying its given "slice" of the grid. All puff information needs to be passed between the master and slave nodes each time step to ensure that each node is up-to-date for the subset of puffs that it needs to work on. This works well for PIGDRIVE.

- The master node is used to initialize and transport all puffs, regardless of their locations within the grid or processor "slices". PIGINIT/PIGWALK presented a problem, given that these routines are called well down inside the nesting routines and work on all puffs throughout the entire domain. Instead of having the master node "skip" the transport/chemistry on all grids (as it had in previous non-PiG CAMx/MPI versions), the master node control needs to flow into EMISTRNS and skip all processes except for the calls to PIGINIT/PIGWALK for the grid step responsible for these processes.

- The master node needs to have the latest wind fields on all grids for this process. Current and updated puff locations need to be received from and sent to all nodes for subsequent PIGDRIVE work. A "barrier" is added just before the PIGINIT call to ensure that all nodes are caught up to the master node, since we do not want the master node to spin off and perform all the puff initialization/walking over the course of an hour as the other nodes are working on the first time step.

The following points address our current thinking on an approach to mitigate potential PiG load balancing issues in follow-on versions of CAMx:

- Load balance problems may arise for PIGDRIVE given that the typical spatial distributions of PiG sources are clumped within specific regions of the domain. Thus, certain nodes operating on those areas would exhibit a large computing burden relative to other nodes. This burden is primarily due to chemistry.

- The leading idea is to further distribute puff work out to other nodes to balance the load. The total number of active puffs would be simply split by the number of nodes and farmed out. A "barrier" will need to be added just before grid chemistry to ensure that all puff information is updated and sent back to the "distributing" nodes.

- PIGDRIVE would need to be re-worked to load vectors of PiG information for chemistry and growth, instead of working on these processes one puff at a time, so that these vectors can be passed to the other nodes for solution.

- It does not appear that any changes to PIGINIT/PIGWALK would be needed.

2.2 CAMx/MPI STATUS

The final PiG approach outlined above has been implemented into CAMx/MPI v4.51 and the model has undergone significant testing and debugging efforts with the help of TCEQ staff. The final version of CAMx/MPI has been delivered to TCEQ, installed on the modeling computer cluster, and is currently undergoing further application testing by TCEQ staff. TCEQ should refer to the CAMx v4.50 User's Guide for general information about CAMx.

ENVIRON is continuing to prepare this code for public release; this additional work is being conducted partially under other contracts and partially under ENVIRON's internal research and development. The official public release of CAMx/MPI is expected during the Fall of 2008. This version will be labeled CAMx v5.0; it will be based on v4.51, with various minor non-MPI updates and bug fixes that have been identified in the standard v4.51 since its release in May 2008. A new User's Guide will accompany its release, complete with instructions on how to configure and run CAMx with MPI (as provided in Section 3 of this report).

2.3 CAMx/MPI SPEED PERFORMANCE

Two cases with very different model configurations were used to test the CAMx/MPI code. The first consists of the standard CAMx test case available for download from www.camx.com: it employs two relatively small grids at 36 and 12 km resolution, centered over St Louis, Missouri. The second was provided by the EPA/OAQPS, and employs two larger grids, the 36-km RPO domain covering most of North America and a 12-km grid covering the entire eastern U.S. Results from both test cases are tabulated below. These results consistently show that MPI is most advantageous for larger applications in which overhead processes (e.g., model setup, I/O, etc.) are a much smaller fraction of total model run time. In other words, CAMx applications on expansive multiple grids, employing extensive chemistry (PM and RTRAC), or including Probing Tools (OSAT, PSAT, DDM) would benefit most from MPI parallization.

2.3.1 Simulation 1

- Grids: 68×68 and 92×115
- 16 layers
- No PiG
- Mechanism 6 (CB05, no PM)

MPI Slices	OMP threads / Slice	Total CPU	Minutes /Sim Day	Equivalent Processors
1	1	1	106	▨
2	1	2	57	1.9
4	1	4	32	3.3
8	1	8	21	5.0
16	1	16	18	5.9
1	4	4	43	2.5
2	4	8	25	4.3
4	4	16	17	6.2

2.3.2 Simulation 2

- Grids: 148×112 and 281×242
- 14 layers
- No PiG
- Mechanism 4 CF (CB4, with PM)

MPI Slices	OMP threads / Slice	Total CPU	Minutes / Sim Day	Equivalent Processors
1	1	1	503	▨
2	1	2	266	1.9
4	1	4	138	3.6
8	1	8	81	6.2
12	1	12	76	7.0
16	1	16	61	8.2
24	1	24	49	10.3
1	8	8	111	4.5
2	8	16	79	6.4
4	4	16	71	7.1
3	8	24	52	9.7
6	5	30*	45	11.2

* "Overloaded" on 24 available processors.

3. INSTALLING AND USING CAMx/MPI

ENVIRON has provided the source code of CAMx/MPI v4.51 developed under this project to the TCEQ. 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 version of CAMx, and may be replaced with revised or alternative instructions upon the final public release version. Information in this section supersedes the installation and compilation instructions provided in Chapter 11 of the CAMx v4.50 User's Guide.

3.1 INSTALLING CAMx/MPI

The CAMx/MPI tar file is compressed. Copy the tar file to the directory to receive the CAMx source code. Issue the following command:

```
tar –xvzf CAMx_v4.51.MPI.PreRelease.src.080831.tar.gz
```

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.

However, 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 de-allocating 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 (v4.51), each compilation of the model results in an executable labeled for a specific application that mirrors the specific `camx.prm` file used to define the static arrays. Since the `camx.prm` file for CAMx/MPI is no longer application specific, we have modified the `Makefile` to produce an executable with the generic name of `CAMx.v4.51.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 specific 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.v4.51.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.v4.51.MPI` directory. Enter the command "`make pg_linux`" or "`make i_linux`". The executable will be automatically named: `CAMx.v4.51.MPI.pg_linux` or `CAMx.v4.51.MPI.i_linux`, depending on which compiler is used. To utilize Open-MP (OMP) in your application, enter the analogous command "`make pg_linuxomp`" or "`make i_linuxomp`". In this case, the executable will be named `CAMx.v4.51.MPI.pg_linuxomp` or `CAMx.v4.51.MPI.i_linuxomp`, and will have the OMP directives for shared-memory parallelization built in.

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. This 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, on the other hand, 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:

   For MPICH1 use the following:

   ```
   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.

   For MPICH2 use the following:

   ```
   cat << ieof > nodes
   10.1.4.2:2
   10.1.4.3:2
   10.1.4.4:2
   10.1.4.5:2
   ieof
   ```

   Notice that the syntax for each entry differs from the MPICH1 version in that the addresses are followed by a colon and a numeric value. This is the convention adopted in

MPICH2 for assigning multiple slices to a single host. Do not list a machine multiple times in the mpd.hosts file.

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

   For MPICH1 use the following:

   ```
   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 `mpich2` 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.

   For MPICH2 use the following:

   ```
   mpdboot -n 1 --file=nodes
   mpiexec -np $numprocs $EXEC
   mpdallexit
   ```

   The `mpdboot` command will initialize the MPD environment with one ring containing the hosts found in the `nodes` file.

7. Implementing Open-MP (OMP) with an MPI application.

   To use a hybrid MPI/OMP approach, follow all of the steps for running an MPI application. In addition, add the environment variable for the number of OMP threads to the CAMx job script.

   For the `bash` shell use:

   ```
   OMP_NUM_THREADS=2
   export OMP_NUM_THREADS
   ```

   For the `csh` shell use:

   ```
   setenv OMP_NUM_THREADS 2
   ```

   This will utilize both MPI and OMP in the simulation. The domain will be divided into slices in the usual way. But when operating on a slice, the host will spawn multiple threads to parallelize the portions of the code where OMP parallelization has been included.