Parallel Acceleration of CALPUFF (Version 5.8 - Level 070623) using MPICH-2

Final Report
Contract No. 11-760

Prepared for:

State of California Air Resources Board
Planning and Technical Support Division
PO Box 2815
Sacramento, CA 95812

Prepared by:

Principal Investigator: Michael J. Kleeman
Graduate Student Researcher: D.J. Rasmussen

Department of Civil and Environmental Engineering
University of California at Davis
One Shields Avenue
Davis, CA 95616
(530) 754-8323

June 17, 2013

Disclaimer

The statements and conclusions in this Report are those of the contractor and not necessarily those of the California Air Resources Board. The mention of commercial products, their source, or their use in connection with material reported herein is not to be construed as actual or implied endorsement of such products.

ACKNOWLEDGEMENTS

The authors are grateful to the California Air Resources Board (Contract No. 11-760) for support of this work.

This Report was submitted in fulfillment of ARB Contract No. 11-760, *Improvements to the Computational Efficiency of the CALPUFF Modeling System (Version 5.8—Level 070623) for Use in Regulatory Applications*, under the sponsorship of the California Air Resources Board. All work was completed as of June 30, 2013.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

ABSTRACT

The Message Passing Interface (MPI) library was used to construct a portable parallel variant of CALPUFF, a source-receptor puff dispersion model, to accelerate numerical calculations. Gridded receptor sampling in the serial CALPUFF model accounts for roughly 99% of the wall clock time during a typical simulation. The parallel version of CALPUFF implements a 1-dimensional decomposition of all sampling receptors across all active processor elements to distribute the workload over multiple compute nodes. A parallel I/O method was also implemented to limit the communication bottlenecks between processors during file writing operations that are known to cause performance degradation. The capabilities of the parallel CALPUFF model are demonstrated in a 90-day simulation with 10,000 area sources, which could not be performed with the current U.S. Environmental Protection Agency (EPA) approved version of serial CALPUFF (version 5.8—level 070623) and currently available computing hardware. Performance improvements between the parallel and serial CALPUFF variants exceed a factor of 16 for simulations with 100 area sources and using 52 processor elements. All results between serial and parallel CALPUFF are found to be equivalent within numerical tolerances, a necessity for continued EPA endorsements. If further performance improvements are needed, the use of a dynamic domain decomposition is suggested to maximize CPU occupancy across all process elements when receptors are not uniformly distributed throughout a domain and/or when receptors are moving within a domain. The parallel CALPUFF model will be a useful tool for regulatory modeling when the number of receptors and/or emission sources is too great to complete a simulation in a reasonable amount of time using the serial CALPUFF model.

EXECUTIVE SUMMARY

Background.

The CALPUFF model tracks emissions from point sources through a simulation of atmospheric dispersion and deposition. The model accounts for time-variation in the emissions rate and meteorological conditions by separately tracking discrete subsets (PUFFs) of the total emissions through the atmospheric simulation. CALPUFF has been used previously to support a number of regulatory decisions worldwide. As the size and complexity of the cutting edge science questions has expanded, the computational limits of CALPUFF have reduced the usefulness of the program. In order to support future regulations, there is a need to optimize the CALPUFF modeling system computational code such that performance is increased.

Methods.

A Fortran profiling tool was used to identify routines and algorithms in CALPUFF that may be candidates for acceleration during representative simulations based on data from past regulatory studies in California. Alternative parallel algorithms that may improve computational efficiency were proposed and implemented. The scalability and performance of the parallel CALPUFF model was tested using varying numbers of emission sources and simulation times. The numerical accuracy of the parallel CALPUFF model was analyzed using appropriate statistical metrics.

Results.

Using scenarios from past regulatory studies in California demonstrates that the sampling of the puffs in the area of the receptors accounts for roughly 99% of the total simulation time in the serial CALPUFF model. Using the MPICH-2 library, a 1-dimensional decomposition of all receptors was implemented across all active processor elements where each process element samples only at its assigned receptors. Parallel output routines were also implemented to eliminate communication bottlenecks that can degrade performance during file writing procedures. For an 8-day simulation with 100 sources, the parallel CALPUFF model is >16 times faster than the serial CALPUFF model with an efficiency of 35-75% when 2-56 processors are used with standard Gbps networking. Furthermore, model simulations encompassing tens of thousands of sources are possible with the parallel CALPUFF model; these problems are not practical using the serial CALPUFF model with commodity hardware at the present time. Concentrations predicted by the parallel CALPUFF model are equivalent to the serial CALPUFF model within numerical tolerances.

Conclusions.

A strategy for computational acceleration of CALPUFF using the MPICH-2 library was proposed and implemented. The parallel CALPUFF model is >16 times faster than the serial CALPUFF model under typical problem sizes. The parallel CALPUFF model has the capability to simulate tens of thousands of emissions sources, which is a problem size beyond the capabilities of the serial CALPUFF model at present time. Predictions from the parallel CALPUFF model are equivalent to predictions from the serial CALPUFF model within numerical tolerances. Future improvements to the parallel CALPUFF model could include a dynamic domain decomposition that would optimize the amount of work that each processor is assigned. The parallel CALPUFF model should be a useful tool for regulatory modeling when the number of receptors and/or emission sources is too great to complete a simulation in a reasonable amount of time using the serial model variant.

CHAPTER I

Introduction

1.1 Introduction

Atmospheric reactive chemical transport models are mathematical representations of pollutant dynamics commonly used on urban (~10^4 km^2) to regional (~10^6 km^2) scales with horizontal resolutions of 4-36km to predict how chemical species concentrations change in response to changes in emissions. These models, frequently referred to as air quality models, are useful in determining and evaluating emissions abatement strategies to comply with air quality standards.

The treatment of atmospheric phenomena in 3-dimentional reactive chemical transport models has evolved considerably over past decades. As the understanding of the underlying chemical and physical processes in the atmosphere continues to grow, there has been a persistent desire to include increased detail in air quality models in pursuit of accurate results that match observed behavior in the atmosphere. These ambitions have proven to be a grand challenge [Levin, 1989] to modelers as computational constraints, notably central processing unit (CPU) speed, have placed limits on certain air quality model parameterizations such as the spatial domain size and horizontal and vertical resolution, simulation length, and the detail of the chemistry and physics that can be included in calculations for simulations to proceed faster than real time [Seinfeld, 1989]. Traditionally, these parameterizations and settings have been simplified to keep calculations tractable. However, many model developers agree that the use of massive parallel computing architecture is a promising solution

to meet the ever-growing computational demands of air quality modeling [Zlatev, 1995].

Previous studies have extensively shown that air quality models are well suited for parallelization [Dabdub and Seinfeld, 1994; 1996; Elbern, 1997; Kumar et al., 1997; Molnr Jr et al., 2010; Saylor and Fernandes, 1993]. Furthermore, the current generation of 3-dimensional gridded Eulerian air quality models (e.g. the Community Multi-scale Air Quality model, or CMAQ; the Weather Research and Forecasting Chemistry model, or WRF-Chem; the Comprehensive Air Quality Model with Extensions, or CAMx) have all demonstrated performance benefits from utilizing massively parallel computing resources [Emery et al., 2008; Grell et al., 2005; Tonse and Brown, 2007].

Gaussian dispersion models (e.g. the American Meteorological Society/EPA Regulatory Model Improvement Committee Model, or AERMOD; the California Puff model, or CALPUFF) represent a subset of the processes that occur in the atmosphere with emphasis placed on advection and diffusion of pollutants subject to simple or no chemical reaction. As such, Gaussian dispersion models are much faster than full air quality models that simulate hundreds to thousands of chemical reactions. Gaussian dispersion models have generally not been rewritten as parallel programs to date. Recently, a goods movement study carried out in California required the simulation of several hundred emissions sources over one year using a Gaussian dispersion model [CARB, 2008]. The computational burden of this problem was too large for a single model application, and so a crude parallel approach was used that involved breaking up the problem into discrete sub-problems that could each be simulated separately. The additional overhead in this manual decomposition resulted in significant effort. The use of parallelism in these popular air quality tools would significantly improve

the amount of detail (e.g. number of emission sources, number of sampling receptors) that can be simulated in a modeling scheme under both a reasonable time allowance and under CPU and memory limitations.

1.2 The CALPUFF modeling system

CALPUFF [Bennett et al., 2002; Scire et al., 2000a], is a multi-layer, multi-species, non-steady-state puff dispersion model that is a component of the CALPUFF modeling system that also includes a meteorology pre-processing program, CALMET, and a post-processing utility, CALPOST. The CALPUFF dispersion model simulates the spatial (3-dimensional) and temporal effects of meteorology on pollutant dispersion, chemical transformation, and removal processes. The chemical conversion mechanism is quasi-linear and includes treatment for SO_2, SO_4^{2-}, NO_x, HNO_3, NO_3^-, NH_3, PM_{10}, $PM_{2.5}$, and inert pollutants. CALPUFF includes a resistance-based dry deposition removal model for gases and particulate matter and uses both scavenging coefficients and precipitation rates to calculate wet deposition. Emission sources may be characterized as point, line, volume, and area sources. The model includes algorithms for simulating the effects of sub-grid scale features such as complex terrain and building downwash. Because of the non-steady state structure of CALPUFF, causal effects and dynamic trajectories can be modeled producing results that are often more realistic than other EPA approved regulatory models, such as AERMOD. The EPA has recommended the use of CALPUFF for source-receptor distances of 50 km to several hundred kilometers with the regulatory default options selected in the CALPUFF control file, but concedes that CALPUFF may be used on a case-by-case basis given that certain criteria are met (see section 3.2 of U.S. EPA [2005]). CALPUFF has been used in numerous studies including those that assess the health effects of emissions reductions [Levy et al., 2002] and to estimate population exposure to power plant effluence [Zhou et al., 2003]. As of December 2012, the EPA recom-

mends the use of CALPUFF version 5.8, level 070623.

CALPUFF produces hourly pollutant concentrations for each modeled species, along with wet and dry deposition fluxes. Concentrations and fluxes are sampled at gridded, non-gridded discrete, and sub-grid scale complex terrain receptors. CALPUFF places no limitations on the maximum number of sources or receptors that can be modeled, and these and other modeled parameters are set at the model compile time. Drastically increasing both the number of emission sources and pollutant receptors for a CALPUFF simulation presents a computational challenge that can be best met with implementing parallel architecture that distributes the models work load over several processing elements.

Previous published attempts to accelerate the computational performance of CALPUFF have used parallel tools such as graphics processing units (GPUs) and the Open Multiprocessing (OpenMP) application program interface to accelerate sampling at only CALPUFFs gridded receptors [Cremades et al., 2010; Suk-Hyun et al., 2011]. However, these attempts have either failed to produce consistent results between sequential and parallel variants [Cremades et al., 2010] or consistency between results was not assessed [Suk-Hyun et al., 2011], a necessity to ensure that EPA model endorsements remain. In this study, performance enhancement is explored by implementing a distributed memory message passing interface (MPI)[Gropp et al., 1994] version of CALPUFF with the goal of producing consistent results between both serial and parallel model variants.

1.3 The Message Passing Interface

The Message Passing Interface - Chameleon 2 (MPICH2) library [Gropp, 2002] is an implementation of the MPI standard (both MPI-1 and MPI-2). Like OpenMP[1], the MPICH2 library allows developers to take advantage of distributed memory across a large number of CPUs. OpenMP is also known to be the least involved method to increase performance for the model developer as the compiler will often solely determine where to perform parallel calculations. With MPICH2, the programmer must decide where to insert the proper subroutines in the code for communication between processing elements. Although this typically makes parallel model development more involved than OpenMP, it allows for greater control over the parallel architecture compared to OpenMP. While start-up hardware costs may be much less, model acceleration using GPUs is the most involved performance improvement option due to the porting of sometimes lengthy code streams to run on the Compute Unified Device Architecture (CUDA) developed by NVIDIA®.

The MPICH2 library is designed in a modular, user-friendly format, and has proven to be a powerful, scalable, and portable performance optimization solution in a variety of science and engineering applications [Jin et al., 2011]. Additionally, the MPICH2 library includes a Fortran interface that pairs well with the CALPUFF modeling system that is written with Fortran77. This eliminates any need to meticulously port the CALPUFF code to take advantage of the MPICH2 library which is composed in the C language. In this report, a methodology is presented for improved computational performance of the EPA recommended version of CALPUFF (version 5.8—level 070623), a multi-layer, multi-species, non-steady-state puff dispersion model, through the implementation of a parallel calculation framework with the MPICH2 library, a message passing interface.

[1] Website for OpenMP tools, `http://www.openmp.org`

In section 2 of this report we introduce a test scenario, profile this simulation, discuss profiling results from the sequential variant of CALPUFF and identify performance bottlenecks that may be candidates for parallelization. We outline and implement a strategy for increased computational performance in section 3. In section 4, we apply our parallel CALPUFF variant to a test case scenario, compare profiling results between the serial and parallel variants, and verify consistency in resultant pollutant concentrations. In section 5 we offer suggestions for future improvement in the computational performance of the parallel variant of CALPUFF. Finally, we conclude our findings in section 6.

CHAPTER II

Profiling

In order to identify performance hurdles within the CALPUFF modeling structure, a dynamic profiling analysis was performed with the PGPROF® profiler. The PGPROF® profiler gathers timing and calling information from the subroutines and functions from the model as it runs. To run the PGPROF® profiler, the model code must be specially compiled with certain profiling flags. Although the data collection during the profiling may add some overhead to the model execution, it is anticipated that the objective of finding the most computationally intensive routines will likely not be hindered due to the added overhead mostly affecting infrequently used functions and routines and the long duration of the profiling time (Section 2.1). It is important to note that profiling results will significantly vary with the manner in which the model is configured to run in the accompanying control file.

2.1 Description of an 8-day test case scenario

In this study, we use a moderately-sized test case simulation with CALPUFF to demonstrate proof-of-concept. Our test simulation uses 3-dimensional gridded hourly weather fields (e.g. wind, temperature), along with 2-dimensional derived hourly fields and variables (e.g. mixing depth, Monin-Obukov length, precipitation rate) from January 1 to December 31 2002 that are prepared by the CALPUFF modeling system meteorology pre-processor, CALMET (v. 5.53a) [Scire et al., 2000b]. For the CALMET simulation, we use a 1092 km $\times$ 1092 km model domain (figure 2.1)

at 4 km × 4 km horizontal resolution with 12 vertical layers specified from the surface (0 m) up to a height of 5 km. This domain is based on Lambert Conformal Conic projection and has been used by ARB for previous modeling simulations [e.g. CARB, 2008]. We use 3-dimensional prognostic meteorology generated over the western US by the fifth-generation Penn State/NCAR mesoscale model (MM5) [Grell et al., 1995], and, when available, use over-land surface (10-m) meteorological data from 279 observation stations. The horizontal resolution of the MM5 prognostic data was 12 km × 12 km. Gridded terrain elevation data are derived from 3-arc second digital elevation models by the United Stated Geological Survey and have a resolution of 90 m × 90 m. We use 14 distinct land-use categories derived from USGS gridded data sets. (For CALMET model control file, see Table 2 of Appendix E1, CARB, 2008)

CALPUFF is used to simulate pollutant dispersion and removal over an 8-day simulation (04/01/2002 00:00 PST to 04/08/2002 23:00 PST) using diesel and diesel-electric powered ocean-going freight vessel emissions data from 2005 represented as 100 area sources [CARB, 2008]. The domain used by the CALPUFF simulation was identical to that of the CALMET domain (figure. 2.1). At the beginning of each hourly time-step, SO_2, NO_x, PM_{10} (in this simulation representing diesel particulate matter), and NH_3 are emitted from each area source as puffs. Pollutant concentrations were sampled each hour at all gridded receptors on a 273 receptor × 273 receptor grid (74,529 total receptors); sampling in this test scenario was not done at discrete, non-gridded or complex terrain receptors. We model SO_2, NO_x, HNO_3, and NH_3 gas chemistry and sulfate, nitrate and PM_{10} aerosol chemistry and physics with the MESOPUFF II chemical mechanism [Scire et al., 1984a; b]. Both wet and dry deposition of particles and gases are modeled, however aqueous-phase transformation of SO_2 and NO_x is not modeled. The ISC-type terrain adjustment method is used and partial plume penetration of elevated inversions is not allowed. The line printer

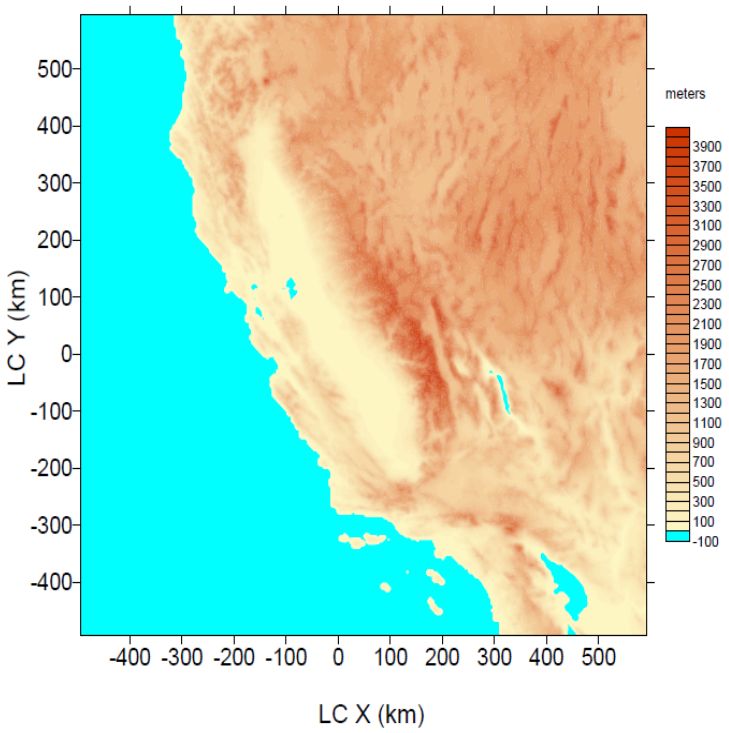

Figure 2.1: The extent of the CALMET and CALPUFF modeling domain used in this study showing terrain elevation (m) derived from 3-arc second digital elevation models by the United Stated Geological Survey with a resolution of 90 m $\times$ 90 m. Lambert Conic Conformal x- and y- coordinates (km) are relative to the domain origin at 37^{o}N 120.5^{o}W. Reprinted with permission from the California Air Resources Board.

output option was turned off to improve model performance. The remaining settings in the CALPUFF control file were configured with the EPA recommended options (see CALPUFF control file in auxiliary material). This 100 source, 8-day test case is chosen because it is computational demanding for the serial CALPUFF variant (˜0.5 hrs. to simulate 1 day), and we hypothesize that a longer profiling duration will yield more accurate timing information from the models subroutines. A variety of weather conditions were present over the modeling domain during this 8-day episode that may make this an interesting, roughly week-long, testing period.

2.2 Profiling results from test scenario

The initial profiling results performed with PGPROF® on an Intel® Core™ 2 Quad Q6600 processor (8MB Cache, 2.40 GHz, 1066 MHz FSB) with 4GB DDR3 SDRAM[2] revealed that the calculations within the subroutine CALCPF accounted for around 99% of the total simulation time (figure 2.2).

Previous studies have profiled the sequential CALPUFF code and have additionally concluded that subroutine CALCPF is the limiting performance hurdle for simulations with the EPA recommended default settings selected in the control file [Cremades et al., 2010; Suk-Hyun et al., 2011]. A brief analysis of the CALPUFF modeling structure (fig. 2.3a) revealed that the subroutine CALCPF lies within the subroutine COMP, a subroutine that encompasses nearly all of the models calculations during each hourly time step. The outer-most loop in the subroutine COMP iterates over each hour of the simulation (fig. 2.3b). For each hour, a loop iterates through each active puff on the computational grid and performs calculations for dispersion, deposition, and chemistry. Further imbedded within the puff loop is subroutine CALCPF, in which a loop is performed over receptors, of gridded, non-gridded discrete, or sub-

[2]Note: All testing and profiling in this report was performed on a 400-node Linux Beowulf cluster maintained by, and located at, the University of California, Davis. Machine specifications are provided in this report when performance results are given.

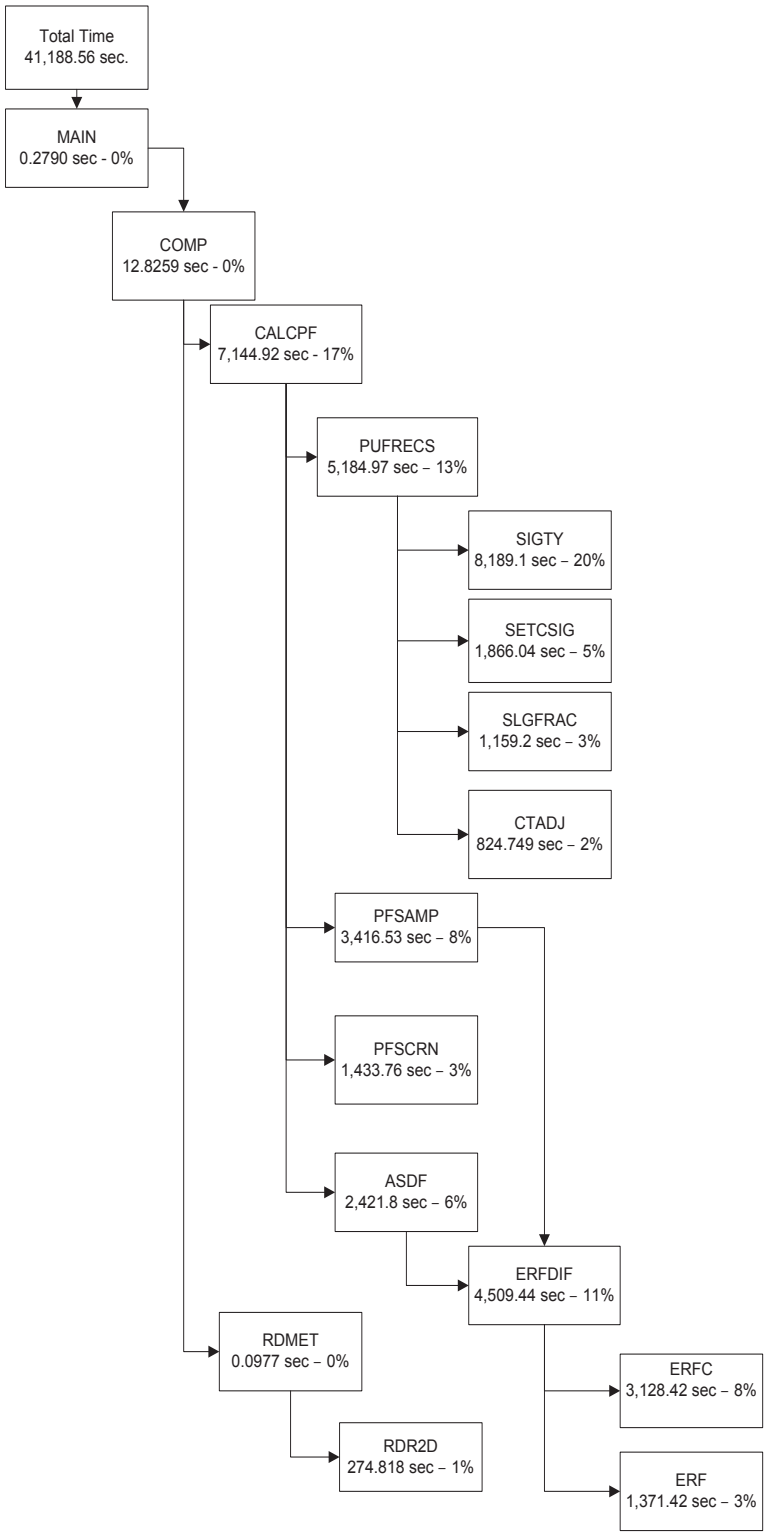

Figure 2.2: A subroutine call tree showing profiling results from an 8-day CALPUFF simulation with 100 area sources with pollutants sampled hourly at 74,529 gridded receptors. Subroutines are only shown if they, or if the subroutines within, contribute at least 1% of the total calculation time. Results obtained with the PGPROF® profiler.

grid scale complex-terrain variety (CTGS) (depending on model setup options in the control file), to sample information about the concentration and deposition fluxes of each species modeled.

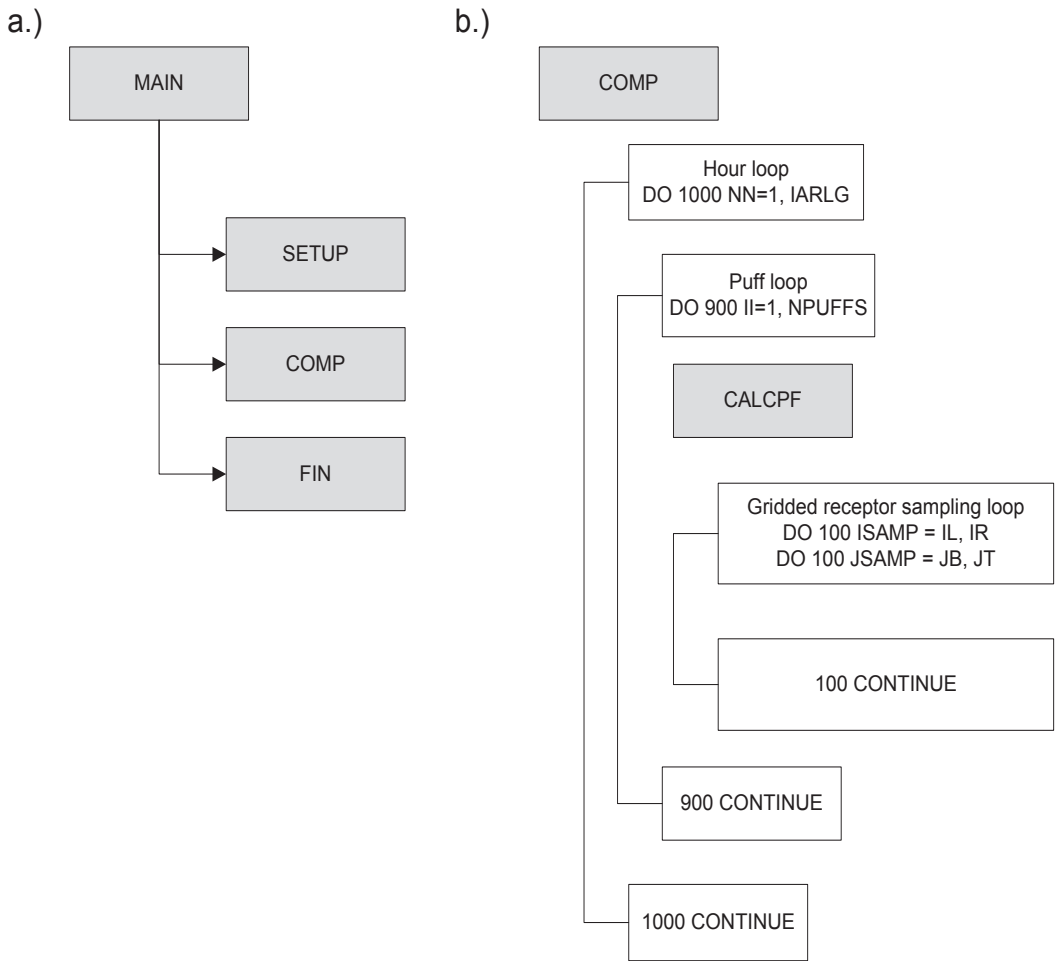

Figure 2.3: a.) The three principal CALPUFF subroutines within the main program body, SETUP, COMP, and FIN; b.) the loop structure within the subroutine COMP that includes the hour and puff loops, as well as the call to subroutine CALCPF and its associated principal loops over all gridded (shown), non-gridded discrete(not shown), and sub-grid scale complex-terrain receptors(not shown).

Closer inspection of the CALCPF code reveals that gridded, discrete, and CTGS receptors are only sampled if they meet certain requirements that render them potentially impacted by the current puff iterate in the puff loop. There is an additional

screening procedure for gridded receptors that further eliminates the need to iterate over all gridded receptors for each active puff. Instead of sampling over all 74,529 receptors in the CA domain, the impacted receptors are identified and then sampled from limits `il` to `ir` and `jb` to `jt` (See Fortran code in figure 2.4a).

a.

```
c variables defined:
c nxsam - number of gridded sampling receptors in x-direction
c nysam - number of gridded sampling receptors in y-direction
c jt - location of the top boundary of the grid affected by the puff
c jb - location of the bottom boundary of the grid affected by the puff
c il - location of the left boundary of the grid affected by the puff
c ir - location of the right boundary of the grid affected by the puff

c --- Loop over gridded receptors (sequential CALPUFF v5.8)
      do 100 irsamp=il,ir
        xr=float(isamp)
          do 100 jsamp=jb,jt
            yr=float(jsamp)
```

b.

```
c variables defined:
c mysamt - top boundary of the 1-dimensional domain decomposition
c mysamb - bottom boundary of the 1-dimensional domain decomposition
```

```
c --- Loop over gridded receptors (parallel CALPUFF)

    if(max(jb,mysamb).GT.min(jt,mysamt)) goto 101

      do 100 irsamp=il,ir

        xr=float(isamp)

        do 100 jsamp=max(jb,mysamb),min(jt,mysamt)

          yr=float(jsamp)
```

Figure 2.4: a) from CALPUFF version 5.8—level 070623; Fortran code that determines if current puff lies over assigned receptors and code that loops over impacted gridded receptors; b) from the parallel CALPUFF variant; Fortran code that determines if current puff lies over assigned receptors and code that determines sampling limits if the latter condition is valid.

CHAPTER III

Parallelization approach

3.1 Implementing a 1-dimensional domain decomposition

Previous studies investigating performance improvements in the dynamic modules of air quality models have proposed differing schemes for disseminating computational workloads amongst processor elements (e.g. [Dabdub and Manohar, 1997; Elbern, 1997; Kumar et al., 1997]). One common approach is to have the root or master process distribute the computational work load to all processors using a predefined 1- or 2- dimensional decomposition of the working domain where each processor element is assigned tasks or calculations to perform. The number of the resultant decompositions is often dependent upon the user-specified number of active processing elements. When using a predefined decomposition, the user will benefit in three ways: (1) the user will know the processing element that corresponds to each taskuseful knowledge when debugging, (2) the user can choose a decomposition where the computational load will be distributed evenly, (3) and the user can select a decomposition where communication overhead is reduced. With each processing element performing calculation for a fraction of the domain, rather than one processor performing all calculations, improvements in computational performance should be realized.

As found in section 2, the subroutine CALCPF, and functions and routines within, are solely responsible for around 99% of the computational load and offers an ideal opportunity to improve model performance while maintaining consistent model results. To achieve this goal, we propose a 1-dimensional decomposition of all active gridded,

non-gridded discrete, and complex terrain sampling receptors across N-active processors. Figure 3.1 diagrams this plan for the test scenario used in this study (section 2.1) where 1-dimensional gridded sampling receptor decomposition is implemented (where each model grid cell is a sampling receptor) using a 273 receptor $\times$ 273 receptor grid that has been divided into four, approximately even, discrete partitions, where each of the four active processors is to sample only the gridded receptors that it has been assigned.

At the start of a parallel CALPUFF simulation, each PE determines the total number of active PEs and then attempts to evenly divide the rows of the sampling receptor domain and the number of discrete receptors by the total number of active PEs. If the number of rows in the sampling receptor domain, or the number of discrete receptors, is not evenly divisible by the number of active PEs, then the root process is assigned the remaining balance of the rows or discrete receptors. Each process uses a common block (`mpidecomp`) to store the upper and lower limit of their assigned gridded receptor rows (`mysamb` and `mysamt`, respectively) and discrete receptors.

When gridded receptor sampling is used, the parallel code in subroutine CALCPF screens for puffs to sample over the processors assigned receptors. If the algorithm finds a puff over its receptors, it will sample (See Fortran code in fig. 2.4b and fig. 3.1). In this approach, all processors will perform the same numerical dispersion, deposition, and chemistry calculations for all active puffs, regardless of the puffs position on the grid. This redundancy may seem computationally inefficient, but with the model settings used in this studys test case, our profiling results (Section 2.2) reveal that non-receptor sampling calculations account for less than 2% of total simulation time. We hypothesize that these redundant calculations can likely occur much faster than if parallelization is implemented because of the known performance penalties

that are associated with communication bandwidth latency, or the start-up time, needed to send and receive messages between processing elements. After each hourly time-step, after all receptors have been sampled for all species and all puffs, species concentrations and wet and dry deposition fluxes from each process are prepared to be written to the disk in binary format.

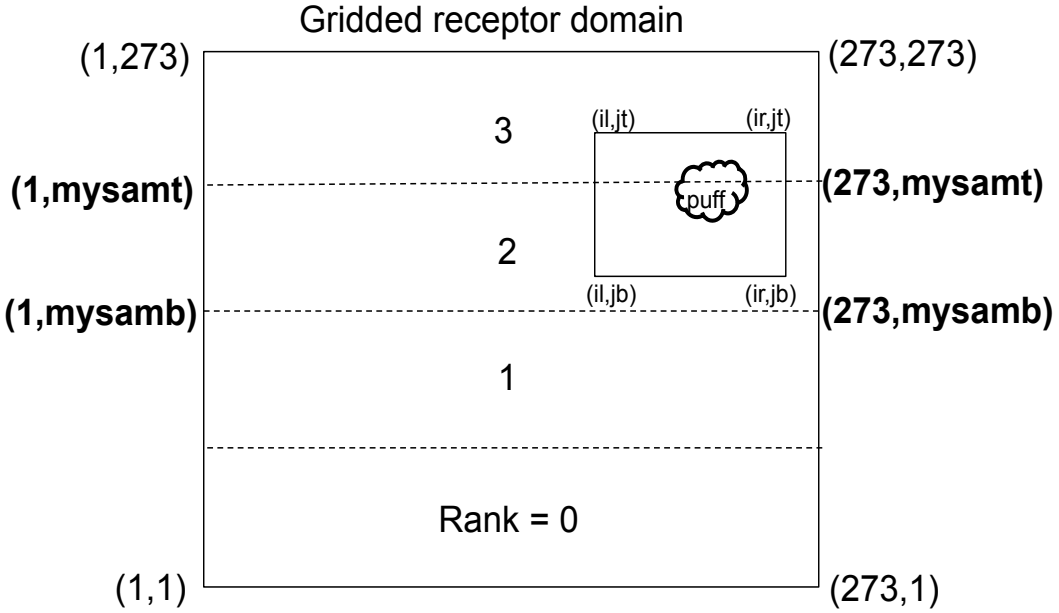

Figure 3.1: A schematic showing a receptor grid of dimension 273 receptors × 273 receptors decomposed 1-dimensionally across four processors, rank 0 through 3. Limits of the receptor grid for the third process element (rank 2) are shown in bold. A hypothetical puff and the associated limits (`il,ir` and `jb, jt`) for the puffs affected receptors (see code in fig. 2.4a) are shown in the northeast corner of the 273 receptor × 273 receptor grid.

3.2 Implementation of parallel I/O routines

3.2.1 Minimizing inter-process communication with parallel output

To minimize or eliminate the need for further communication between processes after the domain decompositions have been assigned, we further propose the implementa-

tion of a parallel output routine where each PE writes its concentrations and fluxes to the disk, rather than passing the data back to the root PE for writing. The EPA recommended version of CALPUFF includes the capability to produce around a dozen output files. However, only writing the species concentrations and wet and dry fluxes require parallel output capability as they are measured at all receptors, which we decompose across all active PEs; all other output files (e.g. restart file, visibility, fog plume, 2-dimensional temperature and density) are redundantly generated by each PE and do not require parallel output. The task of implementing parallel output routines in CALPUFF is largely constrained by the format of the output files produced by the serial version of CALPUFF which are designed to be compatible with the companion CALPUFF post-processing tool, CALPOST (version 6.221—level 082724). All binary output files produced by the parallel variant of CALPUFF must meet the CALPOST formatting requirements.

We use an explicit offset file pointer method (MPI_FILE_WRITE_AT) for parallel output that was revealed in testing to be slightly faster than using a shared file pointer method (MPI_FILE_WRITE) when writing to a local hard disk (Auxiliary figure A.1). Poor write speed performance was experienced with the explicit file pointer method when writing to a redundant array of inexpensive disks (RAID) that did not occur with the shared file pointer method.

The explicit offset file pointer method, by definition, explicitly specifies where each PE should write in the output file. We track the file pointer location using an 8-byte integer variable, `mpifilebytes`, to hold the active write location (bytes), and we update this variable each time writing to the disk occurs. We use an 8-byte integer type because the size of CALPUFF binary output file in bytes may exceed the upper limit of a 4-byte unsigned integer value (2^{32}-1 or ˜4.294×10^{9}). We simulate Fortran block

writing with the MPI parallel output functions in each instance that data is written to disk by first writing the size of the data to be written (in bytes), then writing the data, and then again writing the size of the data that was written (in bytes).

Before concentrations and wet and dry fluxes are sent to the disk for the first time, several headers that contain information about the control file settings used in the CALPUFF simulation are written at the beginning of each binary output file. For simplicity, we conscript the root PE to write these and all other headers and variables. At each hour, both the concentration and flux receptors are sampled in parallel and then prepared to be written to the disk.

Parallelizing data compression routines.

Before concentrations and fluxes are written to the disk, CALPUFF uses a data compression routine to reduce size of the output file. The compression algorithm in CALPUFF packs the concentration and flux fields by replacing sequences of zero values with negative floating point numbers whose absolute value indicates the number of consecutive zeros that have been substituted for the single negative floating point number. For instance:

```
0 0 0 0 1.3E-03 5.2E-03 2.3E-03 0 0
```

becomes

```
-4.0001 1.3E-03 5.2E-03 2.3E-03 -2.0001
```

Because each process only has data from their assigned receptors, all compressed fields must be properly stitched together in the parallel variant of CALPUFF to obtain the full species concentration and deposition fields. To properly reconstruct the global fields, the end points of each compressed field must be communicated so that zero counts may be updated, if necessary. For instance, if compressed field 1 ends with a zero count of 2 (represented as -2.0001) and compressed field 2 begins with a zero

count of 6 (represented as -6.0001), the zero counts must be added (represented as -8.0002) by the PE assigned to field 2. (see below).

Field 1 before updating end point:

```
-4.0001 1.3E-03 5.2E-03 2.3E-03 -2.0001
```

Field 2 before updating end point:

```
-6.0001 4.3E-03 3.3E-03 2.1E-03 -2.0001
```

Field 1 after updating end point:

```
-4.0001 1.3E-03 5.2E-03 2.3E-03
```

Field 2 after updating end point:

```
-8.0002 4.3E-03 3.3E-03 2.1E-03 -2.0001
```

The global fields are properly stitched together when each PE writes its local field to the disk (see Section 3.4). The final form of the entire compressed field must not have consecutive zero counts as they will prohibit proper reading of the simulation results in CALPOST. To facilitate this task of exchanging and updating zero counts between PEs, we use an asynchronous, cascading send/ receive algorithm for all PEs to exchange all concentration and flux field endpoints, with the exception of the first and last PE which exchange just one end point, in subroutine `comprs`.

An attempt to use a collective communication operation (MPI_ALL_GATHER) for the task of exchanging array end points elicited significant network strain using older cluster hardware that would sometimes cause the communication operations to fail when a large number of clustered nodes were used. The cascading send/ receive algorithm has demonstrated good performance when using a larger number of clustered nodes. During testing, both the collective operation and the cascading send/ receive methods were revealed to take approximately the same amount of time to execute.

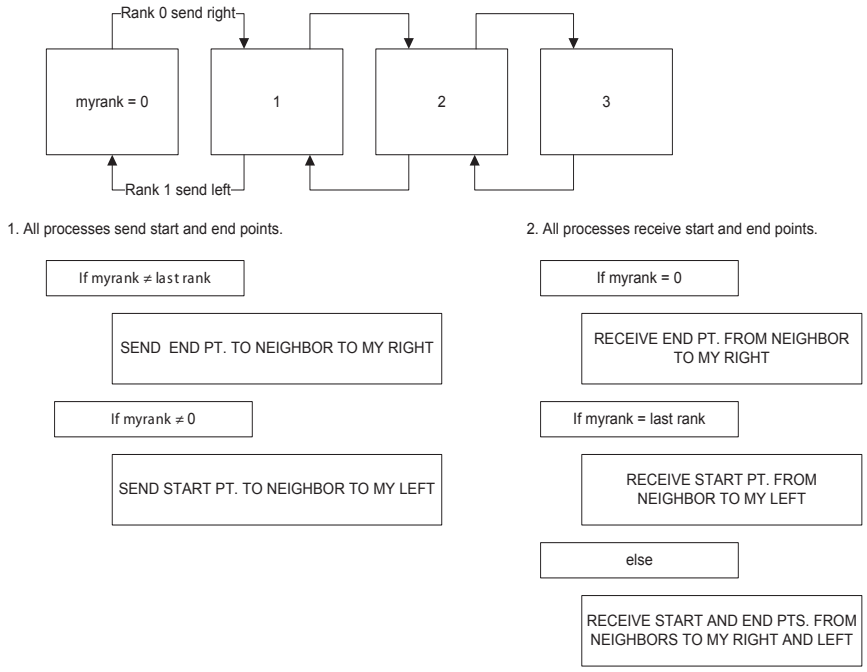

Figure 3.2: Visual depiction of the cascading send/ receive algorithm used in parallel CALPUFF used to exchange array start and end points.

3.2.1.1 Writing compressed concentration and flux fields in parallel

In addition to updating the end points of each PEs local concentration and flux fields, the total number of non-zero and zero values for all global fields must be calculated in subroutine `comprs`. This value is written to disk by the root PE immediately before all other PEs write their concentration and flux fields. CALPOST uses the total number of non-zero and zero values in the algorithm that un-packs the compressed concentration and flux fields. We implement a second cascading send /receive algorithm in subroutine `comprs` to calculate this global value with the root PE (rank = 0) initiating the process by sending its total number of non-zero and zero values to the next PE (rank = 1) where this PE will add its total number of non-zero and zero values to the root PEs count and then send it to the next PE. The sending and receiving continues until all processes have received the counts from their neighbors and have added their counts to the running global sum. The last PE to receive the

count will update its own count and then send the global count back to the root process so that it can write it to disk before all PEs write their local fields.

The running global sum that passes between PEs is also used to assign the file pointer location for each process in preparation for writing out the concentration and flux fields. Extra bytes are added to each processors file pointer location to account for the headers that are written by the root process ahead of the global fields. Before the second cascading send/ receive algorithm, the root PE will pack both the file pointer location and its local count of zero and non-zero concentrations or fluxes into a variable for sending.

Initiate receive all processes wait to get package

```
if myrank.ne.0
 RECEIVE FILE POINTER LOCATION AND LOCAL SUM FROM PROC. TO MY LEFT
 global_sum = mysum + received sum
 file ptr. loc. = received file pointer loc. + header offset
 myoffset = received sum
endif
```

Initiate send

```
if myrank.ne.last rank
  if myrank = 0
    myoffset = 0
    Pkg(1) = mysum
    Pkg(2) = file pointer location
  else
    Pkg(1) = mysum
```

```
    Pkg(2) = received file pointer location
  endif
  SEND PACKAGE TO PROCESS TO MY RIGHT
endif
```

Last processor sends global sum to root process to write to disk

```
if myrank.ne.last rank
   global_sum = mysum
   SEND GLOBAL SUM TO ROOT PROCESS
elseif myrank = 0
   RECEIVE GLOBAL SUM
endif
```

After each PE knows where to write its portion of the global field in the binary file, the process of writing the concentration and flux fields to disk finally occurs. All PEs enter subroutine `wrdatc` and write their portion of the global field using the file pointer location that is calculated from the running sum of the number of zero and non-zero concentrations in each PEs local field. After writing their fields to disk, the PEs return to sample the concentration and flux fields at receptors for the next hour that has been integrated and then they repeat the parallel output process.

3.2.2 Avoiding input file bottlenecks.

At the start of each hour, each PE needs to read meteorological data from the same file at appoximately the same time. When several PEs are in use, bottlenecks can occur during this process, degrading runtime performance. Here, using the `MPI_BROADCAST` function, the root process broadcasts meteorological variables to all active PEs, limiting the number of processes accessing the decreasing the bandwidth required by the file system. This process, along with a summary of the architecture of the parallel CALPUFF variant, is presented in figure 3.3.

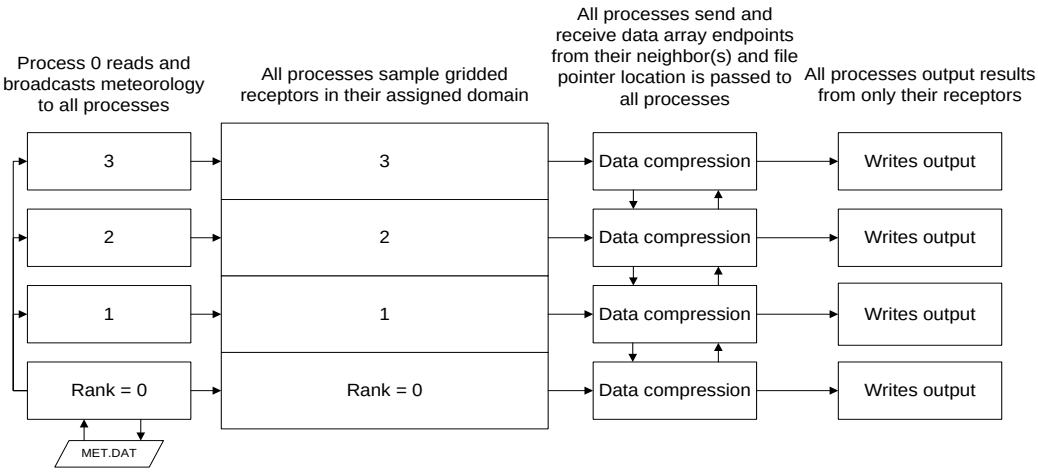

Figure 3.3: Anatomy of parallel CALPUFF gridded receptor sampling and parallel output implementation with a 1-dimensional gridded receptor decomposition across four processor elements.

CHAPTER IV

Performance and scalability of parallel CALPUFF

We perform simulations with both CALPUFF (v5.8) and the parallel CALPUFF models to quantify increases in computational performance, assess scalability, and to assure consistent results between variants. We perform the test case simulation (Section 2.1) using several clustered workstations using the Linux operating system. Running together, these machines create a powerful and flexible distributed memory parallel computer (Linux "Beowulf" system) that is suitable for large air quality modeling calculations. Each node in this analysis, unless otherwise noted, uses an Intel® Core^{TM}2 Quad Q6600 processors (8MB Cache, 2.40 GHz, 1066 MHz FSB) with 4GB DDR3 SDRAM. The network connecting each node was standard Gbps. We vary the number of active PEs to assess computational scaling and efficiency. In addition to the analyzing the performance improvements of running an 8-day test case simulation (Section 2.1) with the parallel CALPUFF variant, we perform 90-day and yearlong test scenarios to gauge resultant speed-ups for bigger problem sizes (i.e. more emissions sources and longer integration times).

4.1 Computational scaling of an 8-day test scenario with the parallel CALPUFF variant

We perform the 8-day simulation (outlined in section 2.1) using the parallel CALPUFF model to asses speeds-ups and scalability. Amdahl's Law relates the expected speed-up to the number of processing elements used and the fraction of the code that can be parallelized:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{4.1}$$

Where S is the expected speed-up factor, P is the fraction of the code that is affected by parallelization, and N is the number of processor cores used.

Our profiling results reveal that around 99% of the sequential CALPUFF code in our 8-day test simulation is confined to the CALCPF subroutine, which we parallelize in section 3.1. Due to this fact, we make the approximation that 99% of the CALPUFF model can be parallelized, and Amdahl's law simplifies to just a function of the number of processor cores used:

$$S(N) = \frac{1}{(0.01) + \frac{0.99}{N}} \tag{4.2}$$

In the upper limit (N$\rightarrow\infty$), Ahmdahl's law theoretically predicts a maximum speed-up of 100. However, to due to overhead associated with communication between processor elements and load imbalances (Section 5), this and other theoretical predictions by Ahmdahl's law are rarely ever fully realized.

Figure 4.1 diagrams the computational scaling for this simulation, indicating the wall clock time (hrs.) to complete the test scenario with 100 area sources as the number of active processors is increased. Additionally plotted is the ideal behavior predicted by Amdahls Law and the computational efficiency (%) defined as:

$$\eta = \frac{Tot.\,Serial\,Time}{Num.\,PEs} \frac{1}{Tot.\,Parallel\,Time} \tag{4.3}$$

We find that computational efficiency drops from 100% with one PE to just above 60% when 4 PEs are used. However, efficiency remains at or just above 50% using between 12 and 52 PEs. Realized speed-ups closely follow the trend in ideal behavior

predicted by Amdahl's law until the number of PEs is greater than 52 and performance degrades. Using 52 PEs, we find that the parallel CALPUFF variant completes the 8-day simulation with 100 area sources over 30 times faster than the serial variant. The optimal number of conscripted PEs will vary depending on the size and type of the problem performed by parallel CALPUFF.

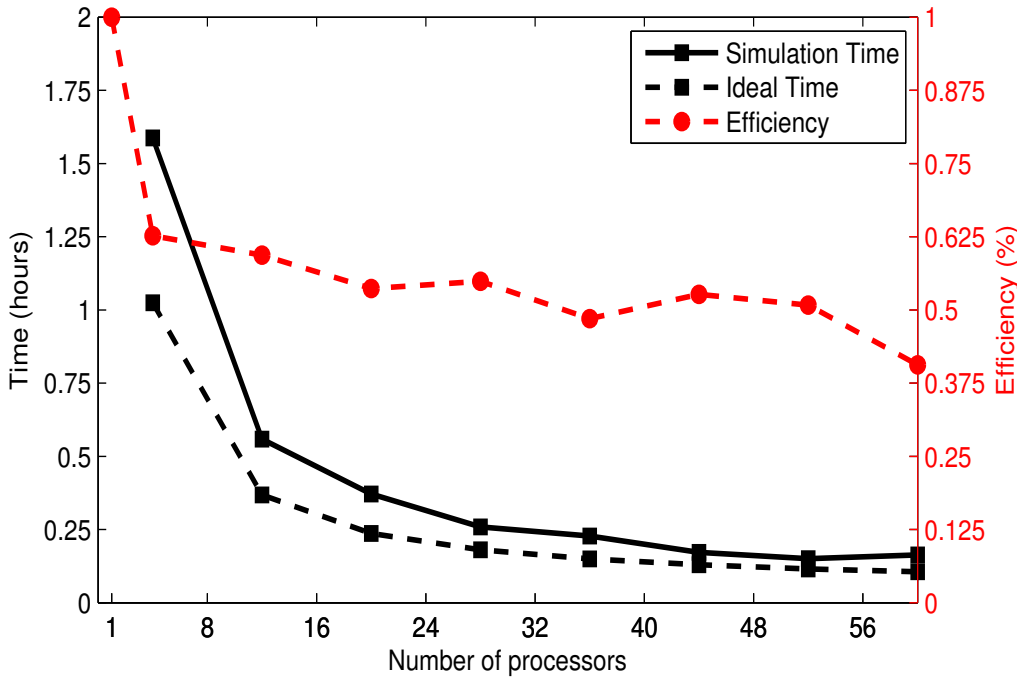

Figure 4.1: Computational scaling of the parallel CALPUFF variant for an 8-day simulation with 100 area sources; a) computation (wall clock) time (hrs.) versus total number of processor cores (solid black line with square markers); b) as for a) but with ideal computation (wall clock) time (hrs.) predicted by Amdahl's Law (dashed black line with square markers); c) as for a) but with computational efficiency (%) versus total number of processor cores (dashed red line with circle markers). Simulations were performed using the settings specified in section 2 and the hardware used was Intel® Core™2 Quad Q6600 processor (8MB Cache, 2.40 GHz, 1066 MHz FSB) with 4GB DDR3 SDRAM. The time to complete the simulation with the sequential model variant was 3.97 hrs. (data point not shown on figure).

4.2 Computational scaling of parallel CALPUFF as the number of sources increases

We find that increases in performance between the serial and parallel CALPUFF variants are highly dependent upon the number of specified emission sources and the length of the simulation time. Although we find a speed up of around 16 from with parallel CALPUFF using 52 clustered nodes for the test simulation specified in section 2.1, improvements in performance will likely be greater for larger problem sizes (i.e. thousands of sources, thousands of receptors, long time integrations).

Using the same 8-day simulation period (04/01/2002 00:00 PST to 04/08/2002 23:00 PST) and 10 clustered nodes, we increase the number of sources to assess the scaling for "larger problem sizes". Figure 4.2 shows the time to complete the 8-day simulation using between 100 and 1000 area sources with an Intel® Core^{TM} i5-3570T (6MB Cache, up to 3.30 GHz) quad-core processors and 8GB DDR3 SDRAM. Figure 4.3 additionally shows profiling results from parallel CALPUFF, but for a 90-day simulation (04/01/2002 00:00 PST to 06/29/2002 23:00 PST) using between 1000 and 10,000 area sources. An additional run of 20,000 area sources was started and simulated two weeks out of 90 days before being stopped due to shifting computing resource priorities on the University of California - Davis Beowulf cluster during the 2012 summer. We feel that a yearlong, simulation involving 20,000 area sources or more, is possible with the parallel variant given that the conscripted computing hardware has at least 8GB of memory to hold several million puffs on the grid at once. Simulations involving tens of thousands of area sources and receptors were all but impractical for the serial variant of CALPUFF, but now may be possible with the parallel model. Because of this, speed-up calculations cannot be practically determined and therefore are not presented.

In both profiling exercises examining the scalability of the parallel variant, input data were read from an Apache™ Hadoop® distributed file system (HDFS). Speed-ups of a factor of two were observed with parallel CALPUFF when the input data were instead read from the network file system (NFS). Therefore the presented results in figures 4.2 and 4.3 are likely a lower end estimates. Computational scaling for simulations using several thousands of sources was not investigated due to computing resource availability during the summer of 2012.

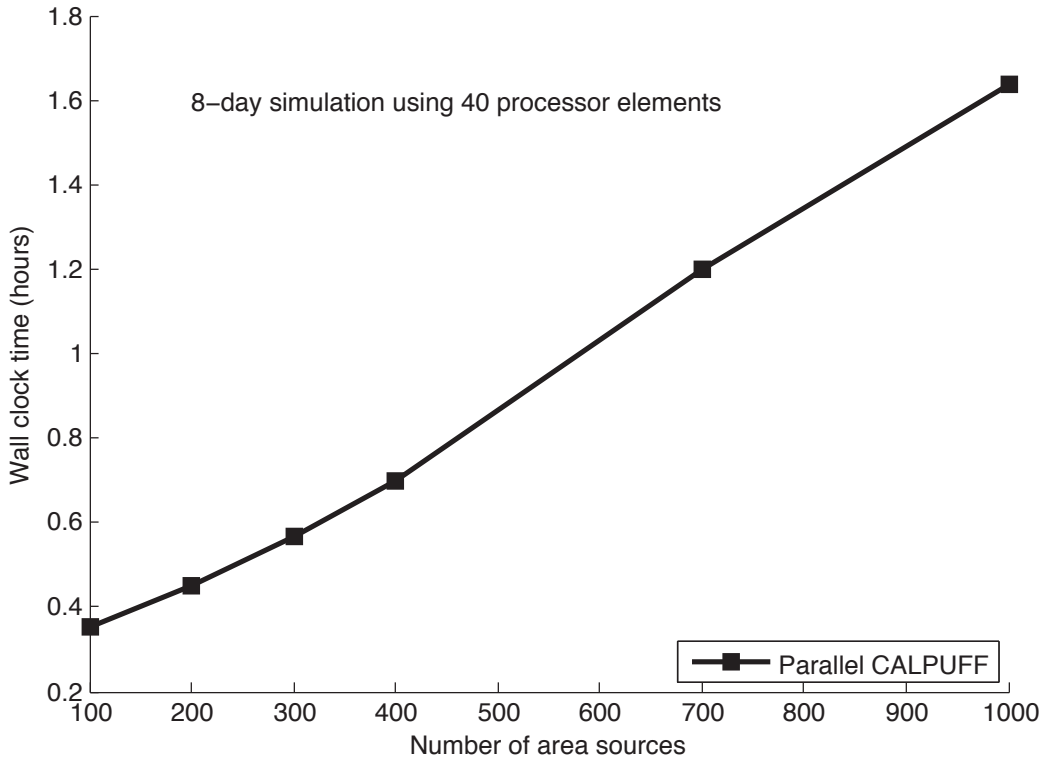

Figure 4.2: Performance of parallel CALPUFF (hours) as a function of the total number of area sources for an 8-day simulation ran on 10 clustered nodes (40 processor elements) with an Intel® Core™ i5-3570T (6MB Cache, up to 3.30 GHz) quad-core processor and 8GB DDR3 SDRAM. Input data were read from an Apache™ Hadoop® distributed file system (HDFS). Speed-ups of a factor of two were observed with parallel CALPUFF when the input data were instead read from the network file system (NFS). Profiling using the serial CALPUFF variant was not attempted.

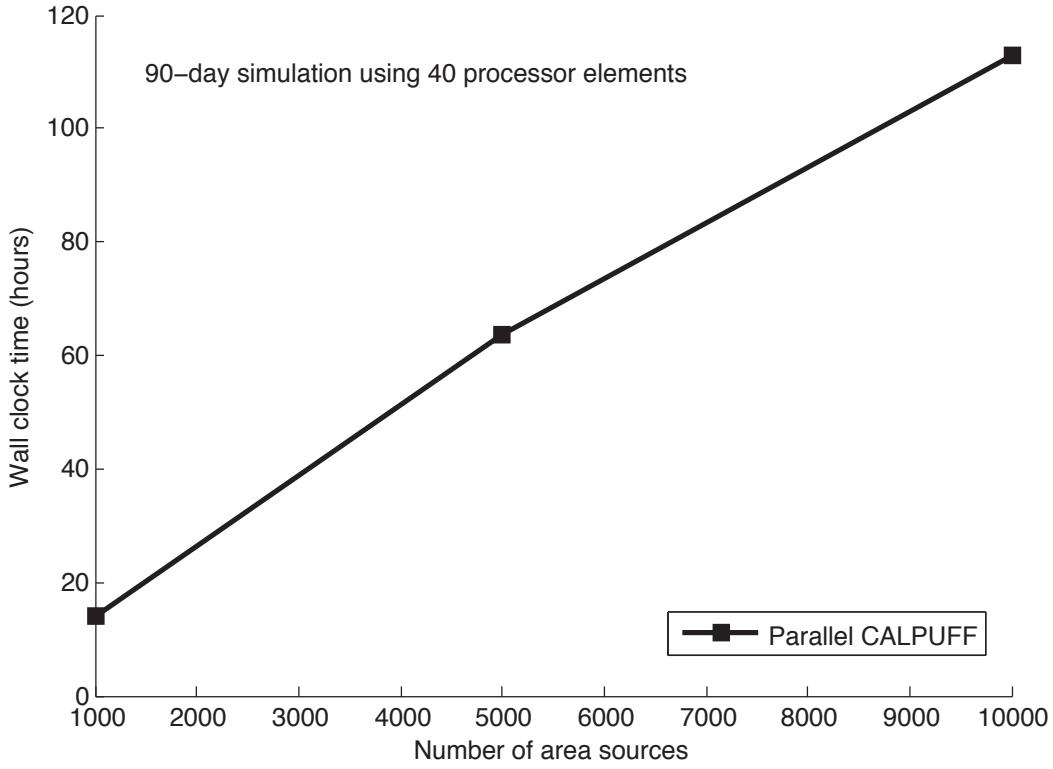

Figure 4.3: Performance of parallel CALPUFF (hours) as a function of the total number of area sources for an 90-day simulation ran on 10 clustered nodes (40 processor elements) with an Intel® Core™ i5-3570T (6MB Cache, up to 3.30 GHz) quad-core processor and 8GB DDR3 SDRAM. Input data were read from an Apache™ Hadoop® distributed file system (HDFS). Speed-ups of a factor of two were observed with parallel CALPUFF when the input data were instead read from the network file system (NFS). Profiling using the serial CALPUFF variant was not attempted.

4.3 Numerical accuracy of parallel CALPUFF

Numerical equivalence in the results for this exercise is paramount for current EPA endorsements to remain, and for this project we adopt the objective of upholding this standard. Previous published attempts to accelerate CALPUFF have either failed to produce consistent results between serial and parallel variants [Cremades et al., 2010] or consistency was not explored [Suk-Hyun et al., 2011].

In this section, we present an additional test case ran with both the serial and paral-

lel CALPUFF variants consisting of a year-long run with 698 area sources to verify numerical accuracy for a moderately sized application that can be simulated using both models in a reasonable amount of wall clock time to allow for a critical evaluation of the numerical results. We present a companion statistical assessment of the numerical accuracy of the parallel model variant relative to the results from the serial model using appropriate metrics. As presented in section 2.1, we model emissions from diesel and diesel-electric powered ocean-going freight vessel emissions off the California Coast. We do not output wet and dry deposition fluxes from the model due to the associated performance penalties.

Concentration fields from the serial variant were generated by splitting up the area sources into 7 groups of 98 or 100 area sources and then simultaneously running each group of emissions sources with the sequential version of the CALPUFF model on seven individual PEs in a "pseudo-parallel" approach. Results from each of the seven simulations were then added together using the CALPUFF system utility CALSUM[3]. This strategy has been a popular method implemented by users of the traditional serial CALPUFF model when dealing with a large number of sources as it would likely take several weeks to complete the simulation if one PE was assigned all 698 sources. We use 15 clustered quad core nodes to perform the simulation with parallel CALPUFF to perform the same simulation scenario. A schematic showing both modeling approaches and simulation times is shown in figure 4.4.

The simulation time given for the sequentially coded variant (fig. 4.4) is the longest amount of time taken to simulate one of the seven concentration fields using one of the seven individual processing elements that is assigned a fraction of the 698 total sources. It should be noted that the simulations performed with the serial

[3]CALSUM source code available at: `http://www.src.com/calpuff/download/mod6_codes.htm`

model were performed on older computing hardware and therefore does not elicit a direct comparison to the time taken for the parallel variant to simulate the same scenario using 15 quad core clustered nodes. As such, we urge caution in judging the performance of the parallel variant solely from this exercise. Additional simulations are needed to accurately gauge speed-ups.

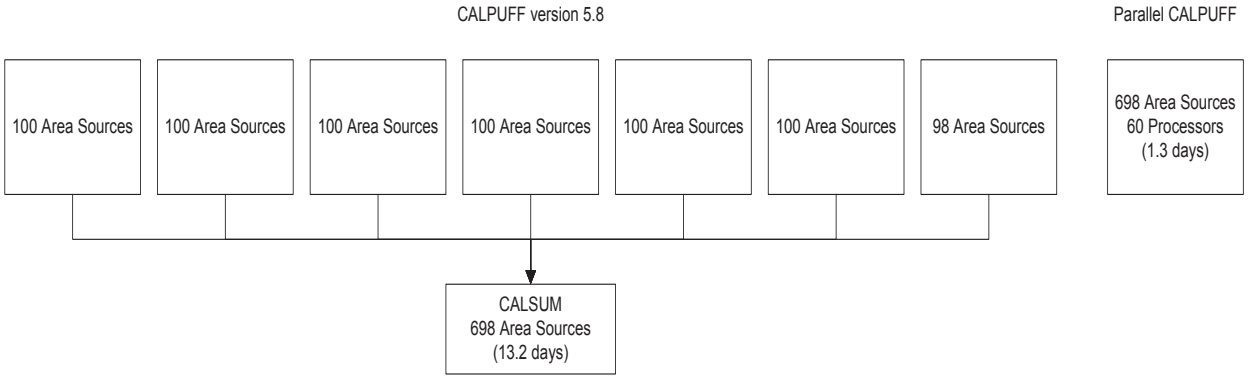

Figure 4.4: Schematic showing approach to using both the serial and parallel variant of CALPUFF with 698 sources. Approach using the serial variant divides up the 698 sources evenly and utilizes seven individual computing resources. The resultant seven concentration fields are then added using the utility, CALSUM. Timing information for the serial variant is presented as the longest amount of time taken to simulate one of the seven concentration fields using one of the seven individual computing resources. The parallel model utilized 60 processor cores across 15 clustered nodes to simulate a yearlong scenario.

A field plot of the annual average PM_{10} concentrations ($\mu g\ m^{-3}$) resultant from the emissions from ocean-going vessels off the coast of California for 2005 is shown in figure 4.5 from both the serial variant (a.) and the parallel variant (b.). The results from the parallel variant appear equivalent to that of the serial model.

More quantitative metrics of model inter-comparison are shown in both figure 4.6 and table 4.1. Each pollutant species (concentrations from the 75^{th} percentile or greater) from CALPUFF is plotted against the serial and parallel variant in figure 4.6. The parallel variant nearly exactly reproduces the results from the serial variant.

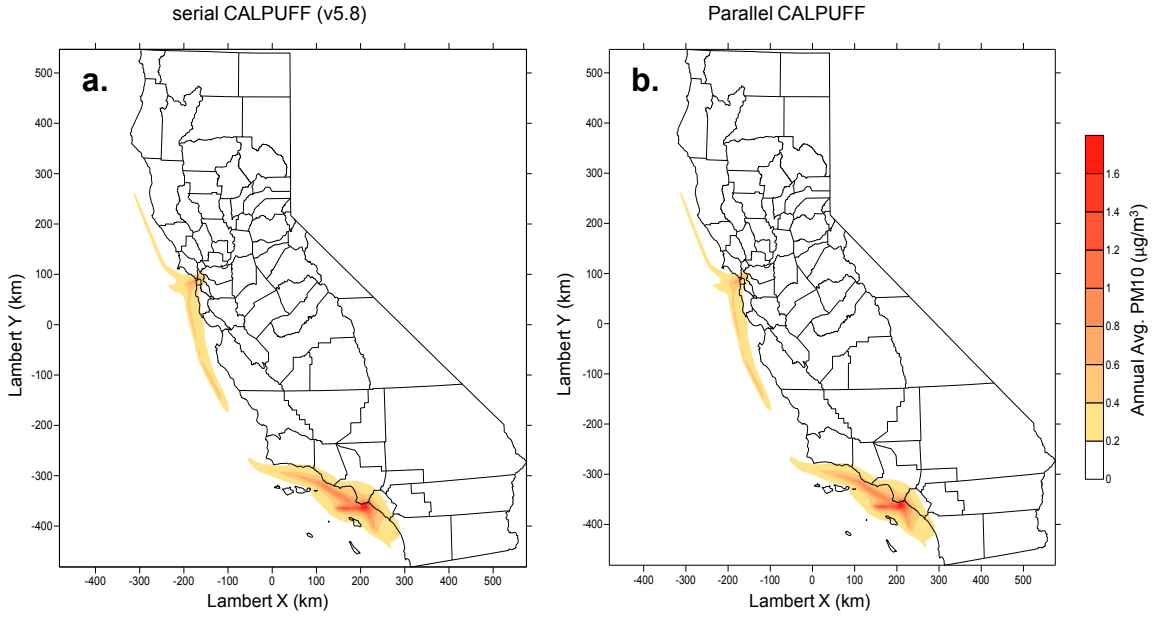

Figure 4.5: Annual average PM_{10} concentrations ($\mu g\ m^{-3}$) resultant from the emissions from ocean-going vessels off the coast of California parameterized in the model as 698 area sources; a) are from the serial CALPUFF variant (version 5.8—level 070623); b) are from parallel CALPUFF. All results are processed with the post-processing software, CALPOST. Parallel CALPUFF results are obtained 10 times faster using 15 cluster nodes than when using the sequential CALPUFF variant (version 5.8—level 070623). Both simulations performed on Intel® Core^{TM}2 Quad Q6600 processors (8MB Cache, 2.40 GHz, 1066 MHz FSB) with 4GB DDR3 SDRAM.

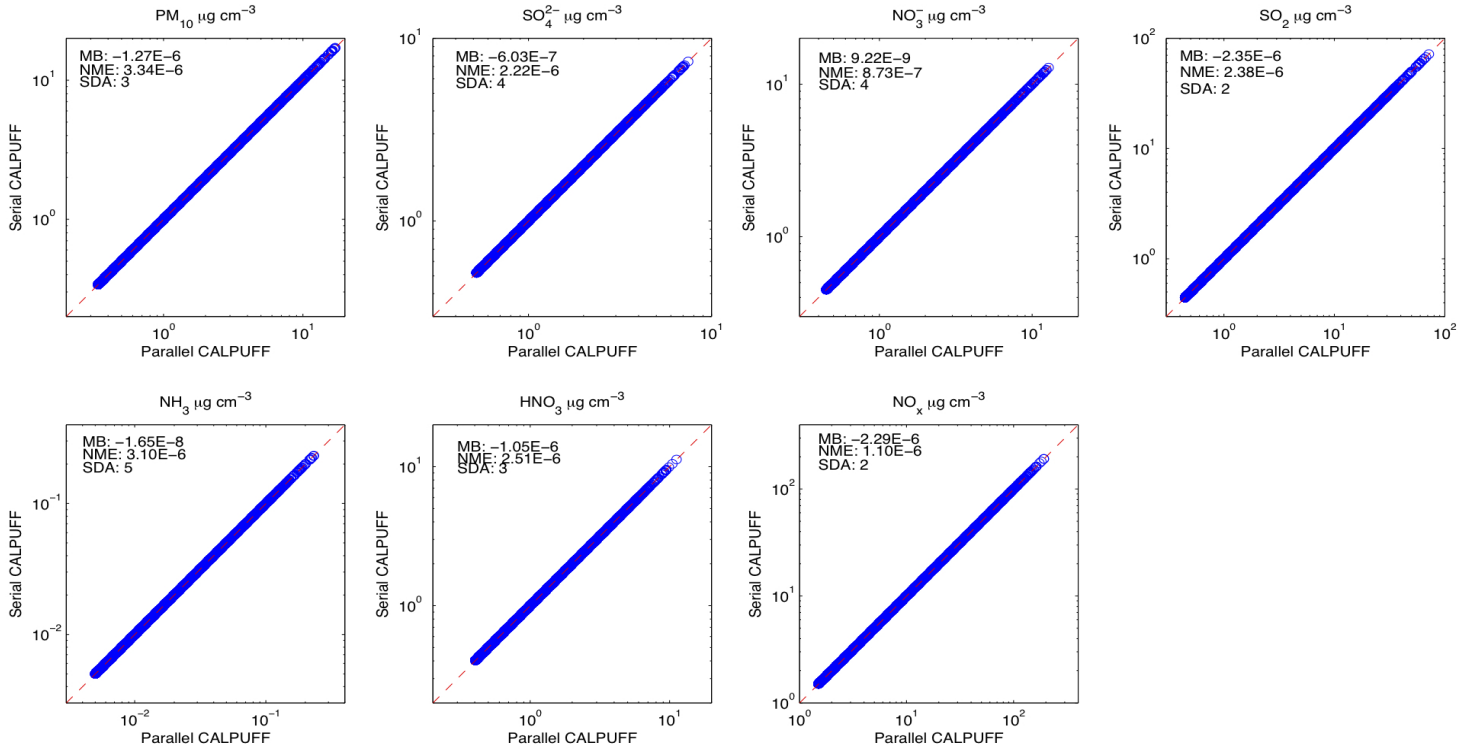

Figure 4.6: Comparison of annual 1-hr average concentrations ($\mu g\ m^{-3}$) at all receptors using emissions from ocean-going vessels from 2005 from a yearlong simulation (described in section 2.1 and section 4.3) performed with serial CALPUFF (version 5.8—level 070623) versus parallel CALPUFF for a.) particulate matter of 10 μm in diameter or less, b.) sulfate, c.) nitrate, d.) sulfur dioxide, e.) ammonia, f.) nitric acid, and g.) oxides of nitrogen ($NO_x = NO_2$+NO). Concentrations (open blue circles) are plotted on a log-log scale. Only species concentrations greater than or equal to the 75^{th} percentile are plotted. The 1:1 line is plotted as a dashed-red line. "MB" is for mean bias, "NME" is normalized mean error, and "SDA" is significant digits of accuracy. Statistics are also listed in table 4.1.

Significant digits of accuracy (SDA) [Sandu et al., 1997] is used to measure the detailed numerical differences between the parallel and serial variant. We calculate SDA using:

$$SDA = -\log\left(max_k \cdot \sqrt{\left(\sum_{C_k,j \geq a} 1\right)^{-1} \cdot \left|\sum_{C_k,j \geq a} \frac{C_{k,j}^{serial} - C_{k,j}}{C_{k,j}^{serial}}\right|^2}\right) \tag{4.4}$$

This metric incorporates a modified root mean square norm of the relative difference of the parallel solution ($C_{k,j}$) with respect to the serial solution ($C_{k,j}^{serial}$) for species k at receptor j. A minimum threshold value, a, prevents inclusion of small concentrations that are not meaningful. We, however, adopt a value of 0 for a to assess differences across all modeled concentrations which span a several orders of magnitude.

Additional metrics used to evaluate numerical accuracy are mean bias, normalized mean bias, and normalized mean error. Results for all model evaluation statistics are presented in table 4.1. The parallel variant is able to reproduce each concentration from 2 to 5 significant digits; however, the biases in all simulated concentrations are negligible, ranging from $1{\times}10^{-6}$ to $1{\times}10^{-9}$ $\mu g\ m^{-3}$.

Species	PM_{10}	SO_4^{2-}	NO_3^-	SO_2	NH_3	HNO_3	NO_x
Mean Bias ($\mu g\ m^{-3}$)	-1.27E-06	-6.03E-07	9.22E-09	-2.35E-06	-1.65E-08	-1.05E-06	-2.29E-06
NMB (%)	-2.73E-04	-1.30E-04	1.97E-06	-1.88E-04	-2.51E-04	-1.91E-04	-7.26E-05
NME (%)	3.34E-04	2.22E-04	8.73E-05	2.38E-04	3.10E-04	2.51E-04	1.10E-04
SDA	3	4	4	2	5	3	2

Table 4.1: Mean bias ($\mu g\ m^{-3}$), normalized mean bias (%), normalized mean error (%), and significant digits of accuracy for each modeled CALPUFF species as an annual average from parallel CALPUFF relative to serial CALPUFF (version 5.8—level 070623) from a yearlong simulation using 698 area sources.

$$Mean\ Bias = \frac{1}{N_{receptors}} \sum_{i=1}^{N} (C_{serial} - C_{parallel}) \tag{4.5}$$

$$Normalized\,Mean\,Bias = \frac{\sum_{i=1}^{N}(C_{serial} - C_{parallel})}{\sum_{i=1}^{N} C_{serial}} \times 100\% \tag{4.6}$$

$$Normalized\,Mean\,Error = \frac{\sum_{i=1}^{N}|C_{serial} - C_{parallel}|}{\sum_{i=1}^{N} C_{serial}} \times 100\% \tag{4.7}$$

CHAPTER V

Recommended future improvements

Although the parallel CALPUFF model has shown promising performance improvements that open new avenues for applications, there are areas for additional speed-ups. Most notably are redundant "puff" calculations performed by each process and load imbalances between processes that may result in underuse of processor elements. Some air quality model grids, like the gridded receptors in CALPUFF, can be dimensioned unequally in the x- and y- directions (i.e. either a square or rectangle) and, depending upon the number of partitions (or active processing elements), may result in a configuration where processors are assigned unequal numbers of grid cells, or receptors, to sample creating load imbalances. Load imbalances may result in processors with smaller workloads waiting for other process elements with greater workloads to finish their tasks. In addition to unequal domain decompositions, instances where puff sampling is unequal (i.e. spatially uneven distribution of emissions sources) may result in a wide range of CPU use. This is the case for the scenario presented in Section 4.3 as the majority of area emissions are confined to the San Francisco Bay and the Los Angeles and Long Beach shipping ports. When the receptor domain is divided up amongst the active PEs, the PEs assigned the receptors near these ports will have the greatest amount of puff sampling to perform.

We define a metric to describe this imbalance result, *CPU occupancy*, defined as the fraction of time each processor spends on receptor sampling during the course of a simulation. A plot of CPU occupancy versus processor element ID is presented in figure 5.1. Processor elements with higher CPU occupancy fractions are in regions where

many puffs must be sampled (e.g. near shipping ports, in this case), whereas processor elements with lower CPU occupancy fractions are in areas where there are fewer emission sources and fewer puffs to sample. We propose using a dynamic workload decomposition, rather than the current static workload (receptor) decomposition, to correct for these imbalances and improve performance.

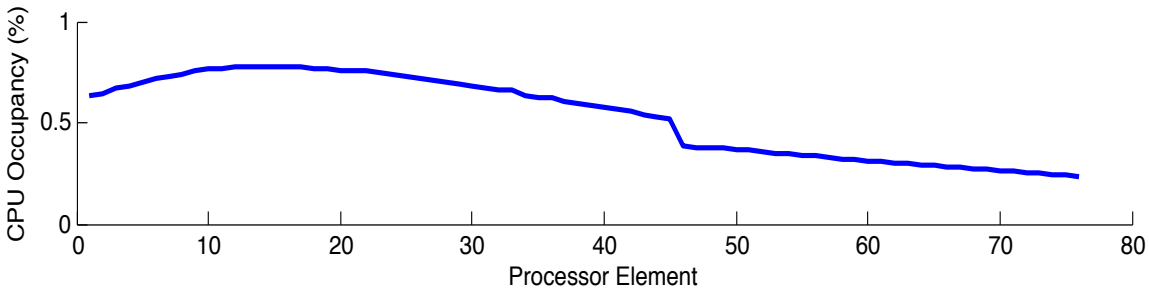

Figure 5.1: CPU occupancy (%), defined as the fraction of time each processor spends on gridded receptor sampling, for each processor element for a yearlong simulation (see Sections 2.1 and 4.3) using 76 processors.

The performance of parallel CALPUFF will improve as the averaging times are increased from the default 1-hour average to the maximum allowed averaging period of 1-year because of the decreased frequency that inter-CPU communication must occur (Section 3.2) before concentration and flux fields are written to binary output. However, users must be careful as the EPA only endorses the hourly concentration output for regulatory applications.

5.1 Redundant "puff" calculations and machine memory limitations

Under high spatial resolution applications of parallel CALPUFF (e.g. California's San Joaquin Valley simulated at 250-m × 250-m resolution with 20,000 sources, and

160,000+ discrete receptors, integrated forward in time for one year), machine memory limitations may arise. The dimensioning of the arrays that hold receptor, puff, and meteorological information in CALPUFF are set at compile time. High spatial resolution applications require these arrays to have substantailly larger dimensions than when the model is used a lower spatial resolution. Sufficient machine memory is required to meet these demands.

A potenital solution to this computational performance hinderance is to free machine memory space for each proccess by avoiding redundant "puff" calculations through fully decomposing the modeling domain's meteorological fields, emission sources, and "puffs" in addition to the already decomposed receptor domain. This future task will be necessary for such high-resolution applications to be possible with out upgrading machine memory hardware (e.g. from 8 GB RAM to 16 GB RAM).

CHAPTER VI

Conclusions

In this report we assess the feasibility of and implement a strategy to improve the computational performance of CALPUFF (version 5.8—level 070623), a regulatory source-receptor air quality model approved by the EPA for long distance air pollution transport. We propose and implement a strategy using the MPICH-2 library. The approach performs a 1-dimensional domain decomposition of all receptors across all active processors. A parallel output routine is also implemented to reduce performance penalties associated with inter-process communication. Speed-ups of 16 over the serial variant are realized, and the capability to simulate tens of thousands of emissions sources is proven, thus allowing for many new applications of the model to be possible. We find equivalent numerical results between the parallel and serial model variants. The biases in all simulated concentrations are negligible, ranging from 1×10^{-6} to 1×10^{-9} $\mu g\ m^{-3}$, and the model is able to reproduce each species concentration from 2 to 5 significant digits. Future improvements to the parallel model that may improve computational performance include fully decomposing the modeling domain's meteorological fields, emission sources, and "puffs". The parallel CALPUFF variant should be a useful tool for regulatory modeling when the number of receptors and/or emission sources is too great to complete a simulation in a reasonable amount of time using the serial model variant.

AUXILLARY MATERIAL A

AUXILLARY MATERIAL

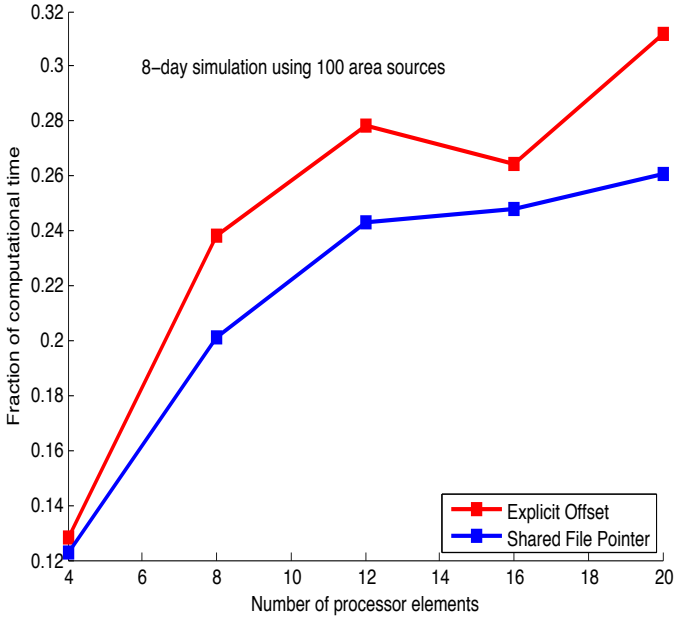

Figure A.1: Scaling of parallel output using an explicit offset file pointer method versus using a shared file pointer method presented as fraction of computational time versus number of processor elements. Results are from an 8-day simulation using 100 area sources.

A.1 Listing of Fortran code additions and modifications

Fortran routine	Purpose
mpif_mod.f90	Tells routine to include 'mpif.h'[%]
mpriranktasks_mod.f90	Hold processor rank element and number of active tasks[%]
mpiwrdatucr.F	Write an uncompressed discrete CTSG receptor concentration/ flux data record in parallel[‡]
mpiwrdatur.F	Write an uncompressed discrete receptor concentration/ flux data record in parallel[‡]
mpilaunch.f90	Launch MPI[%]
mpidecomp_mod.f90	Hold the MPI decomposition variable information[‡]
mpifilesize_mod.f90	Hold the file size in bytes for the explicit offset pointer [‡]
mpiopenlog.f90	Open a log file for each processor element[‡,%]
mpiwrout1.F	Write the header records to output files[‡,#]
mpiopenot.F	Open all input/output files[‡,#]
mpiwrdatu.F	Write an uncompressed gridded concentration or dry/wet flux data record in parallel[‡,#]
mpioutsam.F	Write a gridded field of real or integer numbers in parallel[‡,#]
mpidecomp1D.F	Decompose gridded, non-gridded discrete, and CTSG receptors 1-dimensionally[‡]
mpiwrdatc.F	Write a compressed gridded concentration/ flux data record in parallel[‡,#]
mpioutput.F	Output concentration dry and wet deposition fields in parallel[‡,#]
mpicomprs.F	Compress output array in parallel[‡,#]
mpixtract.F	Extract sampling receptor grid from computational grid[‡,#]
params.puf	Removed hard coded I/O unit numbers for parallel output[‡,#]
CALPUFF.FOR *	Main CALPUFF program[‡,#,%]

Table A.1: *Modified internal subroutines MAIN, SETUP, OPENOT, WROUT1, CALCPF, CALCBC, OUTPUT, COMPRS, FIN, FOGOUT, QAPLOT1, COMP, READCF, RESTARTO, MFLXHDR, MBALHDR, PLMFOG, OUTSAM (‡) Created or modified by D.J. Rasmussen (UC-Davis), (%) Created or modified by Dazhong Yin (CARB), (#) Created by Joe Scire (T.R.C/ Earth Tech)

A.2 CALPUFF test case control file

Variable	Description	EPA Default	Our Values
METDAT	CALMET input data filename	CALMET.DAT	secaq22002.dat
PUFLST	Filename for general output from CALPUFF	CALPUFF.LST	CALPUFF.LST
CONDAT	Filename for output concentration data	CONC.DAT	parallel_test.con
DFDAT	Filename for output dry deposition fluxes	DFLX.DAT	DFLX.DAT
WFDAT	Filename for output wet deposition fluxes	WFLX.DAT	WFLX.DAT
VISDAT	Filename for output relative humidities (for visibility)	VISB.DAT	VISB.DAT
IBYR	Beginning year	User Defined	2002
IBMO	Beginning month	User Defined	4
IBDY	Beginning day	User Defined	1
IBHR	Beginning hour	User Defined	1
IRLG	Length of runs (hours)	User Defined	192
NSPEC	Number of species modeled (for MESOPUFF II chemistry)	User Defined	7
NSE	Number of species emitted	3	4
MRESTART	Restart options (0 = no restart), allows splitting runs into smaller segments	0	0
METFM	Format of input meteorology (1= CALMET)	1	1
AVET	Averaging time lateral dispersion parameters (minutes)	60	60
MGAUSS	Near-field vertical distribution (1 = Gaussian)	1	1
MCTADJ	Terrain adjustments to plume path (3 = Plume path)	3	1
MCTSG	Do we have subgrid hills? (0 = No), allows CTDM-like treatment for subgrid scale hills	0	0
MSLUG	Near-field puff treatment (0 = No slugs)	0	0
MTRANS	Model transitional plume rise? (1 = Yes)	1	1
MTIP	Treat stack tip downwash? (1 = Yes)	1	1
MSHEAR	Treat vertical wind shear? (0 = No)	0	0
MSPLIT	Allow puffs to split? (0 = No)	0	0
MCHEM	MESOPUFF-II Chemistry? (1 = Yes)	1	1
MAQCHEM	Aquesous phase transformation modeled? (0 = No)	0	0
MWET	Model wet deposition? (1 = Yes)	1	1
MDRY	Model dry deposition? (1 = Yes)	1	1
MDISP	Method for dispersion coefficients (3 = PG & MP)	3	3
MTURBVW	Turbulence characterization? (Only if MDISP = 1 or 5)	3	3
MDISP2	Backup coefficients (Only if MDISP = 1 or 5)	3	3

Variable	Description	EPA Default	Our Values
MROUGH	Adjust PG for surface roughness? (0 = No)	0	0
MPARTL	Model partial plume penetration? (0 = No)	1	0
MTINV	Elevated inversion strength (0 = compute from data)	0	0
MPDF	Use PDF for convective dispersion? (0 = No)	0	0
MSGTIBL	Use TIBL module? (0 = No) allows treatment of subgrid scale costal areas	0	0
MREG	Regulatory default checks? (1 = Yes)	1	0
CSPEC	Names of species modeled (for MESOPUF II, must be SO2, SO4, NOx, HNO3, NO3)	User Defined	SO2, SO4, NOX, HNO3, NO3, PM10, NH3
NX	Numer of eas-west grids of input meteorology		273
NY	Number of north-south grids of input meteorology	User Defined	273
NZ	Number of vertical layers of input meteorology	User Defined	12
DGRIDKM	Meteorology grid spacing (km)	User Defined	4
ZFACE	Vertical cell face heights of input meteorology	User Defined	0.,20.,40.,80.,160.,300.,600.,1000., 1500.,2200.,3000.,4000.0,5000.0
XORIGKM	Southwest corner (east-west) of input meteorology	User Defined	-497.132
YORIGIM	Southwest corner (north-south) of input meteorology	User Defined	-494.91
IUTMZN	UTM zone	User Defined	19
XLAT	Latitude of center of meteorology domain	User Defined	37N
XLONG	Longitude of center of meteorology	User Defined	120.5W
XTZ	base time zone of input meteorology	User Defined	PST
IBCOMP	Southwest of X-index of computational domain	User Defined	1
JBCOMP	Southwest of Y-index of computational domain	User Defined	1
IECOMP	Northeast of X-index of computational domain	User Defined	273
JECOMP	Northeast of X-index of computational domain	User Defined	273
LSAMP	Use gridded receptors (T = Yes)	T	T
IBSAMP	Southwest of X-index of receptor grid	User Defined	1
JBSAMP	Southwest of Y-index of receptor grid	User Defined	1
IESAMP	Northeast of X-index of receptor grid	User Defined	273
JESAMP	Northeast of Y-index of receptor grid	User Defined	273
MESHDN	Gridded receptor spacing = DGRIDKM/MESHDN	1	1
ICON	Output concentrations? (1 = Yes)	1	1
IDRY	Output dry deposition flux? (1 = Yes)	1	0
IWET	Output wet deposition flux? (1 = Yes)	1	0
IVIS	Output RH for visibility calculations (1 =Yes)	1	0

Variable	Description	EPA Default	Our Values
LCOMPRS	Use compression option in output? (T = Yes)	T	T
ICPRT	Print concentrations? (0 = No)	0	0
IDPRT	Print dry deposition fluxes? (0 = No)	0	0
IWPRT	Print wet deposition fluxes? (0 = No)	0	0
ICFRQ	Concentrtion print interval (1 = hourly)	1	1
IDFRQ	Dry deposition flux print interval (1 = hourly)	1	1
IWFRQ	Wet deposition flux print interval (1 = hourly)	1	1
IPRTU	Print output units (1 = g/m**3; g/m**2/s)	1	1
IMESG	Status messages to screen (2 = Yes)	2	2
Output Species	Where to output various species	User Defined	All modeled species
LDEBUG	Turn on debug tracking? (F = No)	F	F
Dry Gas Dep.	Chemical parameters of gaseous deposition species	User Defined	SO2, NOX
Dry Part. Dep.	Chemical parameters of particulate deposition species	User Defined	for diesel particulate matter
RCUTR	Reference cutivle resistance (s/cm)	30	30
RGR	Reference ground resistance (s/cm)	10	10
REACTR	Reference reactivity	8	8
NINT	Numer of particle-size intervals	9	9
IVEG	Vegetative state (1 = active and unstressed)	1	1
Wet Dep	Wet deposition parameters	User Defined	for diesel particulate matter
MOZ	Ozone background? (1 = read from ozone.dat)	1	0
BCKO3	Ozone default (ppb) (Use only for missing data)	80	80
BCKNH3	Ammonia background (ppb)	10	10
RNITE1	Nighttime SO2 loss rate (%/hr)	0.2	0.2
RNITE2	Nighttime NOX loss rate (%/hr)	2	2
RNITE3	Nighttime HNO3 loss rate (%/hr)	2	2
SYTDEP	Horizontal size (m) to switch to time dependence	550	550
MHFTSE	Use Heffer for vertical dispersion? (0 = No)	1	1
JSUP	PG Stability class above mixed layer	5	5
CONK1	Stable dispersion constant (Eq. 2.7-3)	0.01	0.01
CONK2	Neutral dispersion constant (Eq. 2.7-4)	0.1	0.1
TBD	Transition for downwash algorithms (0.5 = ISC)	0.5	0.5
IURB1	Beginning urban landuse type	10	10
IURB2	End urban landuse type	19	19

A.3 User Guide to Parallel CALPUFF

1. Requirements

Parallel CALPUFF is written in Fortran and must be compiled with a Fortran compiler, such as the Intel Fortran Compiler (IFORT). The IFORT compiler is free under non-commercial-use licenses, and binary packages are available online for public download.[1]

Parallel CALPUFF must be run in a UNIX environment with the Message Passing Interface - Chameleon (MPICH) library properly installed. MPICH is free, and binary packages are available online for public download. There are currently no existing non-MPI companion algorithm and routines in the parallel CALPUFF code.[2]

2. Compile parallel CALPUFF

Once the aforementioned system requirements have been met, the next step is to un-tar and un-zip the parallel CALPUFF source code.

```
$  tar -zxvf parallel_CALPUFF.tar.gz -C /target_directory
```

In the target directory (the directory that was un-tar'ed) will be a number of files and three directories. The source code in the directory `f90_src` must be compiled before the parallel CALPUFF executable is built. To do this from the target directory:

```
$  cd f90_src
```

...then...

```
$  make
```

[1]IFORT binary packages for UNIX available at: `http://software.intel.com/en-us/non-commercial-software-development`

[2]MPICH binary packages for UNIX available at: `http://www.mpich.org/downloads/`

One the object files in `f90_src` are compiled, the parallel CALPUFF executable may be built in the target directory:

```
$  cd ..
```

...then...

```
$  make
```

If compilation was successful, the parallel CALPUFF executable should now be built. Check to see if it is in the target directory:

```
$  ls calpuff.exe
```

3. Setup CALPUFF list file

Like any CALPUFF simulation, the list or control file needs to have the proper paths to input and output files and the desired model settings adjusted. This is done by editing the input file (*.inp). There is an example input file in the un-tar'ed target directory. The input file must be in the directory with the model data for thees instructions to be successful, see shell script *run_parallel_calpuff.sh*

4. Conscript machines to run parallel CALPUFF

Use the script *farm_to_nodes.sh* to start mpd on the master node (e.g. node 200) and get the mpd port number using:

```
$  ./farm_to_nodes.sh -b 200 200 "mpd -d -e"
```

This should retreive the port number, for instance:

```
mpd_port=51204
```

Determine what nodes will run parallel CALPUFF with the master node (e.g. node 200) and then use the script *farm_to_nodes.sh* and port number to start a ring:

```
$  ./farm_to_nodes.sh -b 191 199  "mpd -d -h n200 -p 51204"
```

This will start a 10 node ring with node 200 as the master node.

5. Execute parallel CALPUFF

Parallel CALPUFF must be executed from the master node. Remote shell to the master node and change directory to where parallel CALPUFF files were unpacked:

```
$  rsh n200
```

```
user@n200 $  cd ../../target_directory
```

The script *run_parallel_calpuff.sh* starts the parallel CALPUFF simulation on the constructed ring of nodes and is executed on the master node in the following manner:

```
user@n200 $ nohup ./run_parallel_calpuff.sh > run.log 2>&1 &
```

This places the simulation in the background and writes both standard output and error messages to a log file. At any time, the trailing contents of the log file can be viewed in real-time by entering:

```
user@n200 $ tail f run.log
```

You may log out and the simulation will continue to run.

BIBILIOGRAPHY

Bennett, M. J., M. E. Yansura, I. G. Hornyik, J. M. Nall, D. G. Caniparoli, and C. G. Ashmore (2002), Evaluation of the CALPUFF Long-range Transport Screening Technique by Comparison to Refined CALPUFF Results for Several Power Plants in Both the Eastern and Western United States., paper presented at Proceedings of the Air & Waste Management Association's 95th Annual Conference, Baltimore, MD, June 23-27, 2002.

CARB (2008), Appendix E1: CALPUFF Dispersion Modeling of Ocean-Going Vessels Emissions, 21.

Cremades, P. G., E. S. Puliafito, and R. P. Fernandez (2010), GPU Acceleration of CALPUFF, Mecnica Computacional, XXIX(71), 7043-7051.

Dabdub, D., and J. H. Seinfeld (1994), Air quality modeling on massively parallel computers, Atmos Environ, 28(9), 1679-1687.

Dabdub, D., and J. H. Seinfeld (1996), Parallel computation in atmospheric chemical modeling, Parallel Comput, 22(1), 111-130.

Dabdub, D., and R. Manohar (1997), Performance and portability of an air quality model, Parallel Comput, 23(14), 2187-2200.

Elbern, H. (1997), Parallelization and load balancing of a comprehensive atmospheric chemistry transport model, Atmos Environ, 31(21), 3561-3574.

Emery, C., G. Wilson, and G. Yarwood (2008), CAMx MULTIPROCESSING CAPABILITY FOR COMPUTER CLUSTERS USING THE MESSAGE PASSING INTERFACE (MPI) PROTOCOLRep., ENVIRON International Corportation.

Grell, G. A., J. Dudhia, and D. R. Stauffer (1995), A description of the fifth-generation Penn State/NCAR mesoscale model (MM5), NCAR Technical Note, NCAR/TN-398+STR, 122pp. Grell, G. A., S. E. Peckham, R. Schmitz, S. A. McKeen, G. Frost, W. C. Skamarock, and B. Eder (2005), Fully coupled online chemistry within the WRF model, Atmos Environ, 39(37), 6957-6975.

Gropp, W. (2002), MPICH2: A New Start for MPI Implementations, in Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, edited, p. 7, Springer-Verlag.

Gropp, W., E. Lusk, and A. Skjellum (1994), Using MPI: Portable Programming with the Message Passing Interface, MIT Press, Cambridge, MA.

Jin, H. Q., D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman (2011), High performance computing using MPI and OpenMP on multi-core parallel systems, Parallel Comput, 37(9), 562-575.

Kumar, N., E. Segall, P. Steenkiste, and A. G. Russell (1997), Parallel and distributed application of an urban-to-regional multiscale model, Computers &; Chemical Engineering, 21(4), 399-408.

Levin, E. (1989), GRAND CHALLENGES TO COMPUTATIONAL SCIENCE, Communications of the Acm, 32(12), 1456-1457.

Levy, J. I., J. D. Spengler, D. Hlinka, D. Sullivan, and D. Moon (2002), Using CALPUFF to evaluate the impacts of power plant emissions in Illinois: model sensitivity and implications, Atmos Environ, 36(6), 1063-1075.

Molnr Jr, F., T. Szakly, R. Mszros, and I. Lagzi (2010), Air pollution modelling using a Graphics Processing Unit with CUDA, Computer Physics Communications, 181(1), 105-112.

Sandu, A., J. G. Verwer, M. Van Loon, G. R. Carmichael, F. A. Potra, D. Dabdub, and J. H. Seinfeld (1997), Benchmarking stiff ode solvers for atmospheric chemistry problems-I. implicit vs explicit, Atmos Environ, 31(19), 3151-3166.

Saylor, R. D., and R. I. Fernandes (1993), On the parallelization of a comprehensive regional-scale air quality model, Atmospheric Environment. Part A. General Topics, 27(4), 625-631.

Scire, J. S., D. G. Strimaitis, and R. J. Yamartino (2000a), A User's Guide for the CALPUFF Dispersion Model (Version 5).

Scire, J. S., F. W. Lurmann, A. Bass, and S. R. Hanna (1984a), User's guide to the MESOPUFF II model and related processor programs. , U.S. Environmental Protection Agency, Research Triangle Park, NC, EPA-600/8-84-013.

Scire, J. S., F. W. Lurmann, A. Bass, and S. R. Hanna (1984b), Development of the MESOPUFF II dispersion model., U.S. Environmental Protection Agency, Research Triangle Park, NC, EPA-600/3-84-057.

Scire, J. S., F. R. Robe, M. E. Fernau, and R. J. Yamartino (2000b), A User's Guide for the CALMET Meteorological Model (Version 5). 332pp.

Seinfeld, J. H. (1989), Urban Air Pollution: State of the Science, Science, 243(4892),

745-752.

Suk-Hyun, Y., Y. Jin-Uk, K. Kyung-Ho, Y. Hee-Young, K. Youn-Seo, and K. Heeyong (2011), CALPUFF module acceleration with OpenMP

Tonse, S. R., and N. J. Brown (2007), Parallel Efficiency Analysis and Performance Improvement of CMAQ V4.5 on a Beowulf Linux ClusterRep. No. 62896, Lawrence Berkeley National Laboratory, Report prepared for California Energy Commission.

U.S. Environmental Protection Agency (2005), Revision to the Guideline on Air Quality Models: Adoption of a Preferred General Purpose (Flat and Complex Terrain) Dispersion Model and Other Revisions; Final RuleRep., 70 FR 68218-68261.

Zhou, Y., J. I. Levy, J. K. Hammitt, and J. S. Evans (2003), Estimating population exposure to power plant emissions using CALPUFF: a case study in Beijing, China, Atmos Environ, 37(6), 815-826.

Zlatev, Z. (1995), Computer Treatment of large air pollution models, 358 pp., Kluwer Academic Publishers, Boston, MA.