Performance and Portability of an Air Quality Model

Donald Dabdub

Department of Mechanical Engineering, University of California at Irvine, Irvine, CA 92697.

Rajit Manohar

Department of Computer Science, California Institute of Technology, Pasadena, CA 91125.

Abstract

We present a portable, parallel implementation of an urban air quality model. The parallel model runs on the Intel Delta, Intel Paragon, IBM SP2, and Cray T3D, using a variety of standard communication libraries. We analyze the performance of the air quality model on these platforms based on a model derived from the parallel communication behavior and sequential execution time of the air quality model. We predict the performance of the next generation air quality models based on this analysis.

1 Introduction

Air quality models (AQMs) predict the spatial and temporal distribution of gaseous species concentrations in the atmosphere. Pollution dynamics is governed by a rich set of physical and chemical phenomenon including advection, turbulent diffusion, chemical transformations, emissions, and deposition. AQMs are mainly used for the evaluation of emission control strategies and planning for the control of air pollution episodes.

The California Institute of Technology (CIT) photochemical model is one such air quality model. It is used to predict the pollution dynamics in the South Coast Air Basin of California. It has also been modified to model pollution in South Korea, Mexico, and a few regions of the U.S.

Environmental modeling is a grand challenge application [17]. In particular, it has been estimated that a 100,000-fold increase in computational requirements will be necessary to incorporate a comprehensive set of atmospheric physics and chemistry into existing air quality models [13].

Most grand challenge applications are limited by the speed of modern supercomputers. To keep the computation tractable, the problem being solved is simplified by omitting certain physical phenomena, or by using less accurate numerical techniques. As more powerful parallel machines become available, computational scientists can incorporate more complex physical phenomena into existing applications. Therefore, they need to know about the limits of modern machines, and about the effect of adding new physics and chemistry parameterizations to existing models on the run-time of the model.

The main objective of this work is to develop a performance model to predict the execution time of a program as a function of the number of processors. The parameters used in the model fall into two categories: machine-dependent or application-dependent. The machine dependent parameters describe the interconnection network and processors. The application dependent parameters describe the inherent parallelism in the program. All the application related parameters can be measured using the sequential version of the application.

We report performance results on different distributed-memory MIMD machines. In particular, the following machines were used to study performance and portability: Intel Touchstone Delta, Intel Paragon XP/S L38, IBM SP2, and Cray T3D.

The air quality model is described in Section 2. Section 3 contains a description of the sequential implementation of the CIT model. Section 4 summarizes the parallelization strategy and describes the interface layer that permits portability. Section 5 derives the performance model used to predict execution time on various architectures. Results of the performance model are compared with actual parallel profiles of the CIT model. Section 6 presents the results of the portable execution, and the issue of load-balancing is discussed. Section 7 discusses some of the requirements of the next generation of air quality models and predicts the computational behavior of the next generation air quality models on supercomputers that are expected to be available in the next two years.

2 The CIT Air Quality Model

The CIT Air Quality Model is a three-dimensional Eulerian urban photochemical model designed to study the dynamics of pollutant transformation and transport in the atmosphere. The underlying equation used in Eulerian air quality models is the atmospheric diffusion equation:

$$\frac{\partial c_i}{\partial t} + \mathbf{u} \cdot \nabla c_i = \nabla \cdot (\mathbf{K} \cdot \nabla c_i) + R_i(\mathbf{c}, T) + S_i(\mathbf{x}, t) \tag{1}$$

where c_i are the elements of the concentration vector $\mathbf{c}$, t is time, $\mathbf{x} = (x, y, z)$ is position, $\mathbf{u} = (u, v, w)$ is the advective flow field, $\mathbf{K}$ is the eddy diffusivity tensor, R_i is the chemical production of species i, T is the temperature, and S_i is the source rate of i.

The CIT model uses a 80×30 rectangular grid in the x, y plane. The computational domain corresponds to an irregular region within this grid composed of 994 columns. Each column corresponds to a 5 km $\times$ 5 km region in the x, y plane and extends 1100 m in height. The columns are discretized into 5 cells in the z direction. Each cell contains 35 gas-phase chemical species. The chemical mechanism contains 106 reactions [19]. Harley *et al.* provide a detailed description of the CIT model [15].

The inherent time scales of the chemical and physical processes of Equation (1) can vary by several orders of magnitude. Such variations pose one of the major challenges in constructing numerical methods for AQMs. Modularity requirements and computational constraints dictate that the processes be split according to their inherent time scales.

Historically, the numerical solution of the atmospheric equation in AQMs has been determined by the most current computational technologies available. With the increasing computational requirements, operator splitting methods have been developed and refined for the solution of AQMs [20]. Splitting methods provide a numerical approach that is both economical and modular.

The basic idea of the splitting process is the sequential use of operators, $\mathcal{L}$, that govern the different phenomena. In the CIT AQM, the horizontal transport operator is decomposed into two separate operators, $\mathcal{L}_x$ and $\mathcal{L}_y$, describing the transport in the x and y directions, respectively. In addition, the chemistry and vertical transport are combined into a single operator $\mathcal{L}_{cz}$.

The solution of the atmospheric diffusion equation in the operator splitting framework is obtained from the following sequence:

$$\mathbf{c}^{t+2\Delta t} = \mathcal{L}_x^{\Delta t}\mathcal{L}_y^{\Delta t}\mathcal{L}_{cz}^{2\Delta t}\mathcal{L}_y^{\Delta t}\mathcal{L}_x^{\Delta t}\mathbf{c}^t \qquad (2)$$

The numerical solution of the horizontal transport operator has been the focus of research of several sub-branches of computational fluid dynamics. In AQMs, the accuracy of the solution of horizontal transport is critical in the presence on nonlinear chemistry [7–10].

The chemistry operator consists of the integration of a system of coupled, nonlinear ordinary differential equations. The development of general purpose integrators for differential equations has received considerable attention [4]. Nevertheless, tailored algorithms have been developed for systems of equations that originate from chemical kinetics [11,24]. In particular, the CIT model uses a hybrid integration algorithm based on a predictor-multi-corrector scheme [28]. The hybrid method uses the fixed space discretization described previously. The reason for this is twofold: first, to keep the problem computationally tractable; second, because of the lack of emissions data for finer space discretization. The time discretization uses a standard adaptive relative error control techniques. Typically the finest time scales occur at dusk and dawn, when the system presents the greatest stiffness due to the influence of sunlight on chemical kinetics.

3 Sequential Profile

The sequential execution time of the CIT air quality model (AQM) in a typical 24-hour simulation of the South Coast Air Basin of California is composed of three major parts. The chemistry and vertical transport computations take 87% of the total time. The computation involves an adaptive time-step integration, and therefore the time taken for the chemistry computation varies across different regions of the computational domain. The transport computation consumes 5% of the total time. Reading the input data files and writing the output requires 7.5% of the total time. The vector of the concentration of all species at every point in the computation domain is written to disk after each hour of simulation time.

The sequential model was executed on a number of different platforms of varying configuration. Table 1 contains a summary of the total time taken by the sequential version of the CIT AQM. The execution time for processors with comparable architectures is proportional to the clock rate of the processor. Note that the CIT AQM uses only 4.5 megabytes of memory at any given time, even though the input data files are substantially larger. Therefore, the performance of the code will not improve by increasing the amount of physical memory in the system. It was found that the Fortran 77 version of the AQM ran approximately 25% faster than its C equivalent on all the architectures shown. We use the Fortran 77 version of the AQM for all the reported timings.

4 Parallel Implementation

Table 2 shows the target parallel machines. Since each of these target architectures is a MIMD machine connected by a message-passing network, the parallel implementation of the AQM is targeted at message-passing architectures. The Intel Delta is the predecessor of the Paragon; a complete description can be found in [16].

4.1 Parallelization strategy

The CIT AQM is parallelized using a "master-worker" strategy. The "master" distributes the work to the available processors and is responsible for reading and writing files. Reading data for the next hour of simulation time is overlapped with the computation for the current time step, thus reducing the I/O time to the time taken to read the input data for the first hour of the simulation.

An examination of the sequential profile shows that the bulk of the time taken by the CIT AQM is spent in the chemistry computation. Therefore, a good parallel implementation of the AQM must be able to effectively distribute the work done in the chemistry computation over a parallel machine. To determine how the computation can be parallelized, we analyze the data-dependencies in the computation itself.

Updating the concentration vector in the chemistry phase at a particular column in the grid can be performed with local information. This part of the computation is parallelized in a straightforward fashion by distributing the different columns among the available processors.

In the general case, the x (y) transport solver needs the value of an entire strip in the x (y) direction of the grid to compute the new concentration at any point in the strip. The horizontal transport operators for the five vertical layers in the computational domain have no data-dependencies. The CIT AQM serves as a testbed for different numerical techniques for solving the transport equation. The aforementioned data-dependencies were assumed to permit easy replacement of the transport solver. For example, knowing that the transport solver uses finite-difference techniques, one could optimize the communication in a parallel implementation; such solver-specific optimizations were not performed. The $\mathcal{L}_x$ operator is parallelized by distributing the concentration vector by rows among the available processors. Rows in different vertical layers might be distributed to different workers in this process. The $\mathcal{L}_y$ operator is parallelized in a similar fashion.

The concentration vector is redistributed between the various phases of the computation. Since each processor has full information about the position of every element of the concentration vector, every processor can send sections of the vector to the appropriate processor for the next phase in the computation; the receiving processor reassembles the vector before proceeding. A detailed description of this parallelization strategy is given by Dabdub and Seinfeld [12].

4.2 Portability

One of the problems with using parallel machines is that vendor-supplied communication primitives have not been standardized. A parallel program can use vendor-supplied subroutine libraries that take full advantage of the underlying hardware. However, using system-dependent libraries affects portability.

An alternative approach is to use communication libraries that have system-dependent implementations but provide a uniform interface that is system-independent. Commonly used libraries that fall into this category include PVM, MPI, and p4 [3,22,25]. Although using these libraries provides a degree of portability, they were not available on all platforms. In addition, the performance of the primitives provided by these libraries on the parallel machines varies considerably.

The approach used was to develop an interface that was based on the common features of the system-independent libraries and implement it using a particular library on a given system. Since all the libraries provided primitives for sending and receiving arrays of bytes, blocking sends and receives were used as the basic primitives for communication. It was assumed that each process in a computation is assigned a unique identifier from 0 to $N - 1$, where N is the number of processes in the computation, and messages can be sent by any process to any other process. This model is similar to the basic model used by p4, MPI, PVM, and Intel's NX library. Each interface function can be implemented using one or two function calls in the underlying parallel communication library. In fact, the implementation of the communication primitives usually consisted of calling a library function with a permuted version of the arguments! The result was a clean interface to a parallel machine with very little overhead beyond that introduced by the underlying library. This simple interface layer can be used to write other parallel applications as well. The interface permitted us to port the model not only to different parallel machines, but also to use different parallel libraries with minimal effort. Table 2 shows the communication libraries used on various parallel machines.

The air quality model was written to be independent of the number of available processors so that it could be run not only on different machines, but also on the same machine when other users were using part of the machine. Finally, the use of a parallel file system was avoided since the interfaces to parallel file systems are highly system-dependent.

5 Performance Analysis

The performance of a parallel program on a MIMD machine depends on the architectural details of the machine. It is assumed that the parallel machine consists of a number of computational nodes that communicate with one another by the exchange of messages. In this section we develop a performance model for the parallel implementation of the CIT AQM using techniques similar to those described in [14].

Each processor in the parallel machine performs part of the computation required by the AQM. Since the processors operate in parallel, the total time taken by the computation is the maximum of the times taken by the individual processors.

The time taken by a single processor can be logically divided into three parts corresponding to what the processor may be doing at any given instant:

- T_{compute}: The time spent by processor i for computation.
- T_{comm}: The time spent by processor i for communication with other processors because of data-dependencies in the computation.
- T_{idle}: The time spent by processor i when waiting for information from another processor in the form of a message.

The computation time T^i_{compute} is determined by the speed of the processing node i. The communication time T^i_{comm} is determined by the speed of the interconnection network. The idle time T^i_{idle} depends on the communication patterns in the parallel algorithm. The time taken by a single processor i is the sum of these three components.

The computation time is estimated by executing the *sequential* CIT model on the processing node. The total time taken by the sequential program, T_{seq}, consists of the time taken for computation, T_{compute} (92.5%), and of the time taken for reading and writing files, T_{io} (7.5%). A profile of the sequential code serves as the basis for determining this time. Table 1 provides a list of T_{seq} for a variety of architectures. The parallel implementation of the CIT AQM uses a special host node that performs all of the I/O operations on behalf of the worker nodes. Given N worker nodes in the computation, the computation time is estimated to be:

$$T^i_{\text{compute}} = 92.5\% \times \frac{T_{\text{seq}}}{N} \tag{3}$$

The idle time is a measure of the amount of time that was “wasted” during the computation. When the computational work is not evenly divided among the processors, this time can be large. For simplicity, we assume that the idle time is small (i.e., zero) for the performance model.

When a message is sent from one processor to another, there is a constant overhead in time, the latency L, that is incurred because the first data item in the message must travel a fixed distance to the remote processor. Each additional data item is delivered to the remote processor at a rate known as the bandwidth BW (measured in bytes per second). The communication time is therefore estimated by counting the number of messages that are sent and the size of those messages. The observed bandwidth of an interconnection network depends on the length of the messages being sent. Since most of the messages sent during the computation were approximately 1000 bytes long, the bandwidth of interest is the bandwidth of the network for messages that are approximately 1000 bytes in length.

The communication patterns of the CIT model are independent of the underlying interconnect topology, which is partly why the model is portable. Two processors that need to communicate may not be directly connected by a dedicated communication link in the underlying hardware. The result is that two messages that are logically unrelated may be ordered by the underlying hardware, which would cause the communication to be slower than expected. It is assumed that this effect is negligible for the purposes of the performance model.

The latency overhead is encountered for each message sent. For simplicity, it is assumed that the latency for communicating between processors is a constant, independent of the physical location of the processors. The communication time for a particular processor i is given by:

$$T^i_{\text{comm}} = L \times (\text{number of messages}) + \frac{\text{number of bytes sent}}{BW} \tag{4}$$

where the number of messages and the number of bytes sent are calculated for a single worker node, not the entire computation. Both of these quantities depend on the number of processors. The time can be written in more detail as follows:

$$T^i_{\text{comm}} = L \times aN + \frac{bN + c + d/N}{BW} \tag{5}$$

where a, b, c, and d are determined by the sizes of the arrays that are part of the CIT AQM data. The number of messages that are sent by a single processor increases linearly with the number of workers, corresponding to the term aN. The constant b corresponds to data that is replicated at each worker; c corresponds to data that is evenly divided among all the workers; d corresponds to data that is evenly divided which is exchanged by processors in parallel.

The total time taken by a worker is given by the following relation:

$$T = 92.5\% \times \frac{T_{\text{seq}}}{N} + L \times aN + \frac{bN + c + d/N}{BW} \tag{6}$$

All of the parameters of this model can be determined without executing the parallel version of the CIT AQM. The sequential execution time is measured by running the sequential version of the CIT AQM. The parameters a, b, c, and d are obtained by textual inspection of the parallel program, since the entire communication behavior is known a priori.

This simple performance model agrees well with observed behavior. Figure 1 shows both the observed and predicted times for the CIT model on the Intel Delta, Intel Paragon, Cray T3D, and IBM SP2 respectively. For small numbers of processors, the CIT model scales well. As the number of processors increases, the overhead of communication outweighs the advantages of distributing the computation over a larger number of worker nodes.

6 Results and discussion

Figure 2 shows the total time taken by the CIT air quality model during a typical 24-hour simulation of the South Coast Air Basin of California on each of the architectures shown in Table 2. The measured times correspond to wall-clock time and include the time for process creation. The best performance is obtained when using 64 processors of the Cray T3D. Under this configuration, the entire model runs in 341 seconds. As predicted by the performance model, the communication overhead overshadows the advantages of distributing the computation as the number of processors increases.

Figure 3 shows the predicted computation and communication times on an IBM SP2. Observe that in the regime of small numbers of processors, the computation time dominates the communication time. As the number of processors increases, the communication time initially decreases because each individual node sends a smaller amount of data and the nodes communicate simultaneously. Eventually the cost of distributing the replicated data overshadows the advantages of distributing the computation.

One of the problems with any parallel implementation of the CIT AQM is that the chemistry computation uses adaptive integration to solve the chemical kinetic equations. As a result, the time taken for this part of the computation can vary considerably between different vertical columns in the computational domain.

If the chemistry operator was perfectly load-balanced, the time taken by the parallel chemistry computation would be the time taken by the sequential chemistry computation divided by the number of workers. However, in reality, the time taken by the different workers will vary, and the actual time taken by the parallel chemistry computation is given by the time taken by the slowest worker. The load imbalance is the difference in these two times, and varies with the number of workers.

Figure 3 shows the load imbalance as a function of the number of processors. As the number of processors increases, the load imbalance decreases. To reduce the imbalance would entail dynamically load-balancing the computation. Introducing dynamic load-balancing would significantly complicate the implementation and would cause different processors to send an unequal number of messages between the phases in the computation, thus unbalancing the communication phase.

We are interested in improving the accuracy of the CIT AQM, and our next step will be to incorporate the aerosol phase into the parallel model. Figure 3 shows that beyond 25 processors the load imbalance drops to under 30 seconds on an IBM SP2. Since this is not a large amount of real time, the load-balancing issue has been postponed until the aerosol phase is incorporated into the air quality model. Including the aerosol phase will change the load imbalance as well, and as of now it is unclear whether load-balancing the new model will be worthwhile.

7 The Future of Parallel Air Quality Models

Historically, the state-of-the-art for AQMs has been determined by the computational technologies available. Box models were one of the first approaches to model air quality. They consisted, as their name implies, of a single box or cell that encompassed the entire region to be modeled. The main use of box models was to predict the temporal regional average of pollutant concentrations. Most of the computations carried out by box models consist of the solution of chemical reactions of pollutants in the cell. To study air quality on a more detailed scale, trajectory models were developed.

Trajectory models are based on the solution of the atmospheric reaction diffusion equation (Equation (1)) using a Lagrangian coordinate system [18]. Trajectory models are approximately 3-5 times more computationally intensive than box models. To model the spatial and temporal variations of pollutant dynamics over an entire region, grid models were developed.

Grid models are based on the solution of the atmospheric reaction diffusion equation using an Eulerian coordinate system. The region to be modeled is similar to that of the box model. However, the space is subdivided into a three-dimensional array of grid cells. Grid models are approximately 5000 times more computationally intensive than box models.

Eulerian gas-phase air quality models started to evolve in the early 1980's to incorporate gas/aerosol equilibrium [2]. Pilinis and Seinfeld incorporated the aerosol dynamics into a gas-phase three-dimensional comprehensive urban airshed model [23]. The model was able to predict size-dependent aerosol concentration of particles in the South Coast Air Basin of California. This effort tripled the computational demands imposed by the model.

The first generation of aerosol models assumed thermodynamic equilibrium between the gas and the aerosol phases for the volatile compounds. Experimental measurements have reported departures from equilibrium between gas and aerosol phase of the volatile compounds [1,26]. Furthermore, theoretical studies by Meng and Seinfeld have shown that the equilibrium assumption cannot be established under many conditions [21]. However, the main reason for adopting the equilibrium assumption is a lack of computer power. A fully detailed aerosol dynamic calculation requires approximately 50 times the computer time needed to model only the Eulerian gas-phase dynamics. The next generation of air quality models is expected to incorporate both the aerosol and aqueous phase.

The performance model can predict the behavior of the CIT AQM on machines that do not exist yet! Incorporation of the aerosol phase is expected to increase the time taken by the chemistry phase by at least a factor of 50. Our model predicts that, given the preliminary information available on the machines that will be available in the next two years, our current implementation of the CIT AQM will scale to about 500 processors on the new parallel machines such as the Intel ASCI Red teraflop machine. Despite increasing the computational demands, it is expected that the CIT AQM with the aerosol phase will require about 10 minutes of execution time on machines that will be available in the next two years.

8 Conclusions

This paper presents a portable, parallel implementation of the CIT air quality model. By using a simple system-independent interface to parallel libraries, the core of the model was made portable across existing parallel libraries. This permitted the code to be run on a number of parallel machines without modification. The advantage of portability does not hinder performance. The paper describes a performance model used to predict the execution time of the code on different current architectures, as well as architectures that are expected to appear in the near future. To our knowledge, this is the first time that parallel performance results of air quality models have been presented on different computing platforms using a single portable implementation.

The research community is currently actively involved in incorporating aerosol and aqueous phases into existing gas-phase air quality models. Using the performance model developed here, results indicate that the current portable parallel version of the CIT model with the aerosol physics will take about 10 minutes on supercomputers due out in the next two years.

References

[1] Allen A. G., Harrison R. M. and Erisman J. Field measurements of the dissociation of ammonium nitrate and ammonium chloride aerosols. *Atmospheric Environment* **23,** 1591–1599 (1989).

[2] Bassett M. and Seinfeld J. H. Atmospheric equilibrium model of sulfate and nitrate aerosols. *Atmospheric Environment* **17,** 2237–2252 (1983).

[3] Butler R. M. and Lusk E. L. Monitors, messages, and clusters—the p4 parallel programming system. *Parallel Computing* **20,** 547–564 (1994).

[4] Byrne G. D. and Hindmarsh A. C. Stiff ODE solvers: a review of current and coming attractions. *J. Comp. Phys.* **70,** 1–62 (1987).

[5] Carmichael G. R., Peters L. K. and Kitada T. A second generation model for regional-scale transport/chemistry/deposition. *Atmospheric Environment* **20,** 173–188 (1986).

[6] Chang J. S., Brost R. A., Isaksen I. S. A., Madronich S., Middleton P., Stockwell W. R. and Walcek C. J. A three-dimensional Eulerian acid deposition model: physical concepts and formulation. *J. Geophys. Res.* **92,** 14681–14700 (1987).

[7] Chock D. P. and Dunker A. M. A comparison of numerical methods for solving the advection equation. *Atmospheric Environment* **17,** 11–24 (1983).

[8] Chock D. P. A comparison of numerical methods for solving the advection equation—II. *Atmospheric Environment* **19,** 571–586 (1985).

[9] Chock D. P. A comparison of numerical methods for solving the advection equation—III. *Atmospheric Environment* **25A,** 853–871 (1991).

[10] Dabdub D. and Seinfeld J. H. Numerical advective schemes used in air quality models—sequential and parallel implementation. *Atmospheric Environment* **28,** 3369-3385 (1994).

[11] Dabdub D. and Seinfeld J. H. Extrapolation techniques used in the solution of stiff ODEs associated with chemical kinetics of air quality models. *Atmospheric Environment* **29,** 403-410 (1995).

[12] Dabdub D. and Seinfeld J. H. Parallel computation in atmospheric chemical modeling. *Parallel Computing* **22,** 111–130 (1996).

[13] Dennis R. L., Byun D. W., Novak J. H., Galluppi K. J., Coats C. J., and Vouk, MȦ. The Next Generation of Integrated Air Quality Modeling—EPA's Models-3. *Atmospheric Environment*, **30,** 1925–1938 (1996).

[14] Foster, I. T. *Designing and Building Parallel Programs.* Addison-Wesley, 1995.

[15] Harley R. A., Russell A. G., McRae G. J., Cass G. R. and Seinfeld, J. H. Photochemical modeling of the Southern California air quality study. *Environ. Sci. Technol.* **27,** 378–388 (1993).

[16] Intel Corporation. *A Touchstone DELTA system description.* Intel Supercomputer Systems Division, February 1991.

[17] Levin E. Grand Challenges to computational science. *Comm. ACM.* **32,** 1456-1457.

[18] Liu M. K. and Seinfeld J. H. On the validity of grid and trajectory models of urban air pollution. *Atmospheric Environment* **9,** 555 (1975).

[19] Lurmann F. W., Carter W. P. L., and Coyner L. A. A surrogate species chemical reaction mechanism for urban-scale air quality simulation models. EPA No. 68-02-4104 (1987).

[20] McRae G. J., Goodin W. R. and Seinfeld J. H. Numerical solution of the atmospheric diffusion equation for chemically reactive flows. *J. Comp. Phys.* **45,** 1–42 (1982).

[21] Meng Z. and Seinfeld J. H. Time scales to achieve atmospheric gas-aerosol equilibrium for volatile species. *Atmospheric Environment* **30,** 2889–2900 (1996).

[22] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing* **8** (1994).

[23] Pilinis C. and Seinfeld J. H. Development and evaluation of an Eulerian photochemical gas-aerosol model. *Atmospheric Environment* **22** 1985–2001 (1988).

[24] Sandu A., Verwer J. G., van Loon M., Carmichael G. R., Potra F. A., Dabdub D. and Seinfeld J. H. Benchmarking stiff ODE solvers for atmospheric chemistry problems I: implicit versus explicit. *Atmospheric Environment.* To appear.

[25] Sunderam V. S. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience* **2,** 315–339 (1990).

[26] Tanner R. L. An ambient experimental study of phase equilibrium in the atmosphere system aerosol $H^+, NH_4^+, SO_4^{2-}, NO_3^- - -NH_3$(g), HNO_3(g). *Atmospheric Environment* **16,** 2935–2942 (1982).

[27] Venkatram A., Karamchandani P. K. and Misra P. K. Testing a comprehensive acid deposition model. *Atmospheric Environment* **22,** 737–747 (1988).

[28] Young T. R. and Boris J. P. A numerical technique for solving stiff ordinary differential equations associated with the chemical kinetics of reactive-flow problems. *J. Phys. Chem.* **81,** 2424-2427 (1977).

Machine	**Configuration**	**Time (sec)**
Touchstone Delta	16 MB memory; 4KB inst. cache 8KB data cache 40 MHz Intel i860 processor	31529
Intel Paragon XP/S	32 MB memory; 16KB inst. cache 16KB data cache 50 MHz Intel i860 XP processor	30468
SGI Indy	64 MB memory; 16KB inst. cache 16KB data cache; 512KB level 2 cache 133 MHz R4600 processor	7561
Sun Sparc 20	32 MB memory; 20KB inst. cache 16KB data cache 50 MHz SuperSPARC processor	7442
SGI Indy	256 MB two way interleaved memory 16KB inst. cache; 16KB data cache 1MB level 2 cache 100MHz R4400 processor	5719
IBM RISC 580	32 MB memory; 32KB inst. cache 64KB data cache 62 MHz POWER processor	4684
IBM RISC 370	32 MB memory; 32KB inst. cache 32KB data cache 62 MHz POWER processor	4607
SGI	288 MB memory; 16 KB inst. cache 16KB data cache; 1MB level 2 cache 150MHz R4400 processor	3813
IBM RISC 390	32 MB memory; 32KB inst. cache 32KB data cache; 1MB level 2 cache 67 MHz POWER2 processor	3535
SGI Power Onyx	512 MB two way interleaved memory 16KB inst. cache; 16KB data cache 4MB level 2 cache 75 MHz R8000 processor	2823
DEC AlphaStation 600	128 MB memory; 16KB inst. cache 16KB data cache; 96KB level 2 cache 2MB level 3 cache 266 MHz Alpha 21164 processor	2502
Intel Pentium Pro	64 MB memory; 8KB inst. cache 8KB data cache; 256KB level 2 cache 200 MHz Intel Pentium Pro processor	2070

Table 1. Performance of the sequential CIT AQM on various machines.

Machine	**Processing Node**	**Network**	**Communication Library**
Intel Delta	40MHz i860	2D mesh	NX
Intel Paragon	50MHz i860 XP	2D mesh	NX
Cray T3D	150MHz Alpha 21064	3D torus	PVM
IBM SP2	62MHz POWER	omega switch	p4

Table 2. Parallel architectures used.

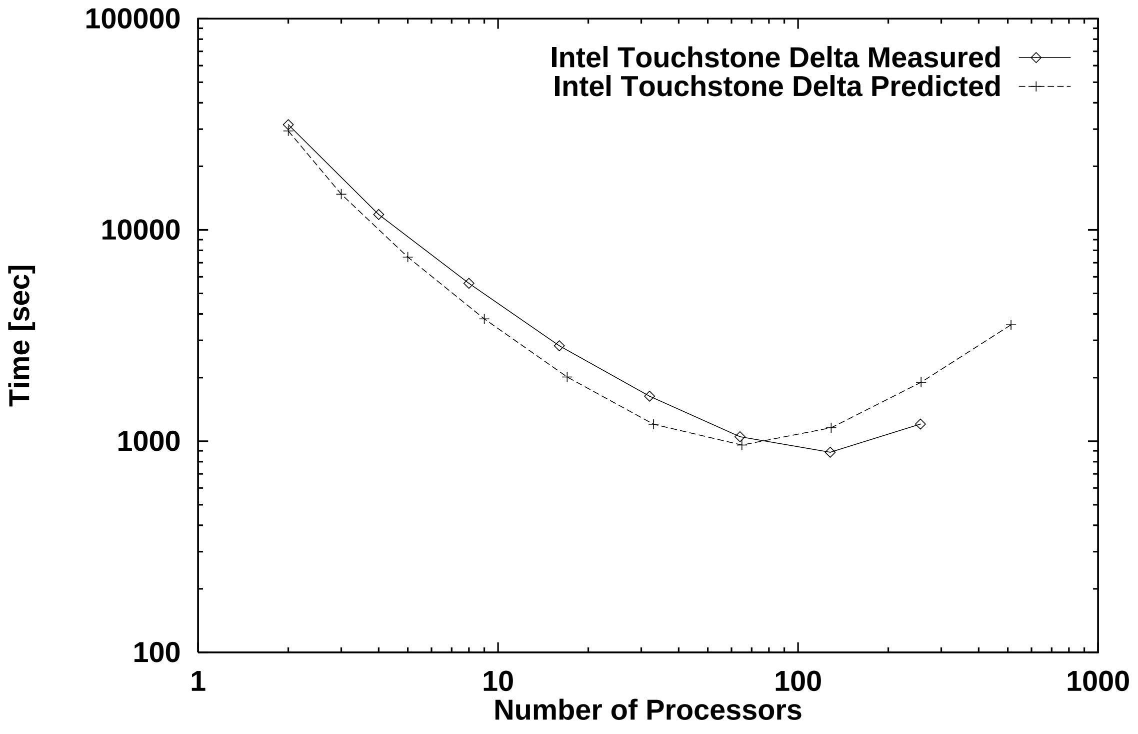

Figure 1a. Actual versus predicted wall-clock time for the CIT AQM.

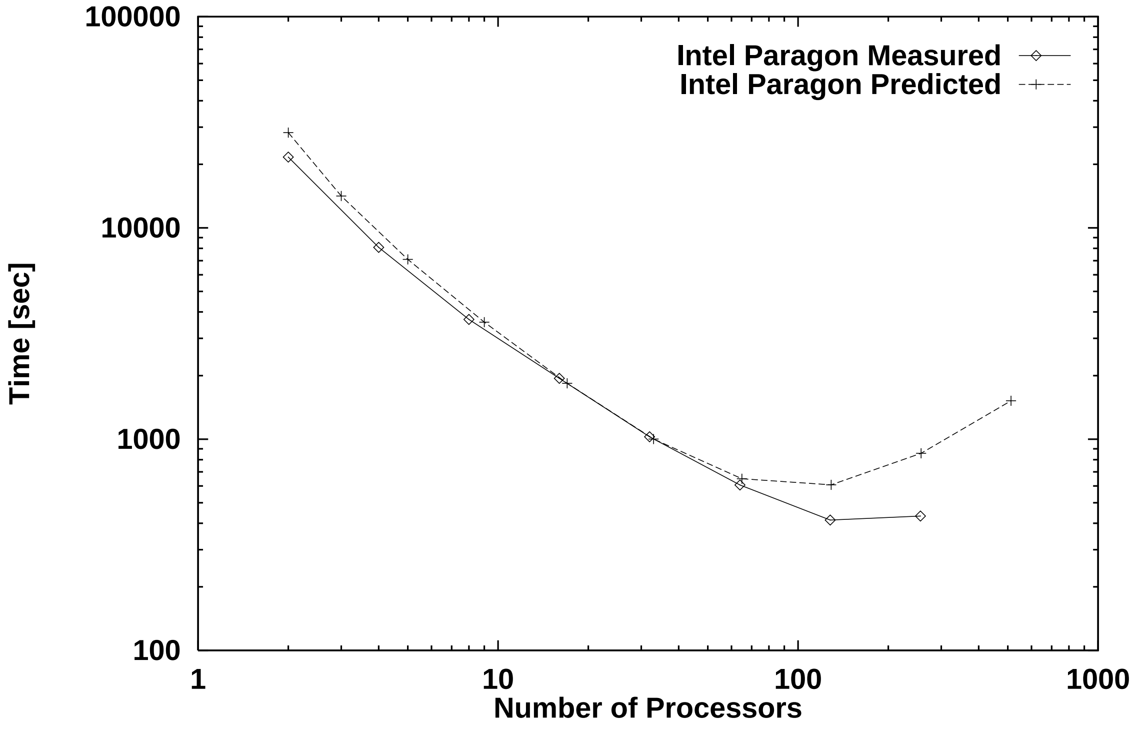

Figure 1b. Actual versus predicted wall-clock time for the CIT AQM.

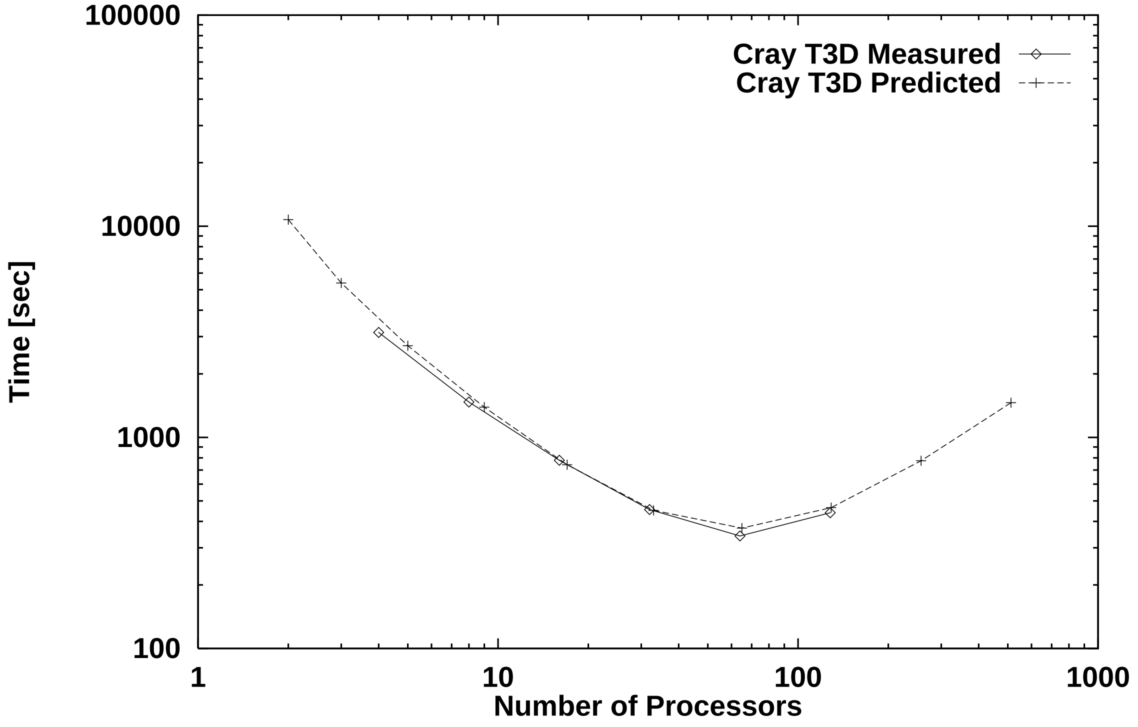

Figure 1c. Actual versus predicted wall-clock time for the CIT AQM.

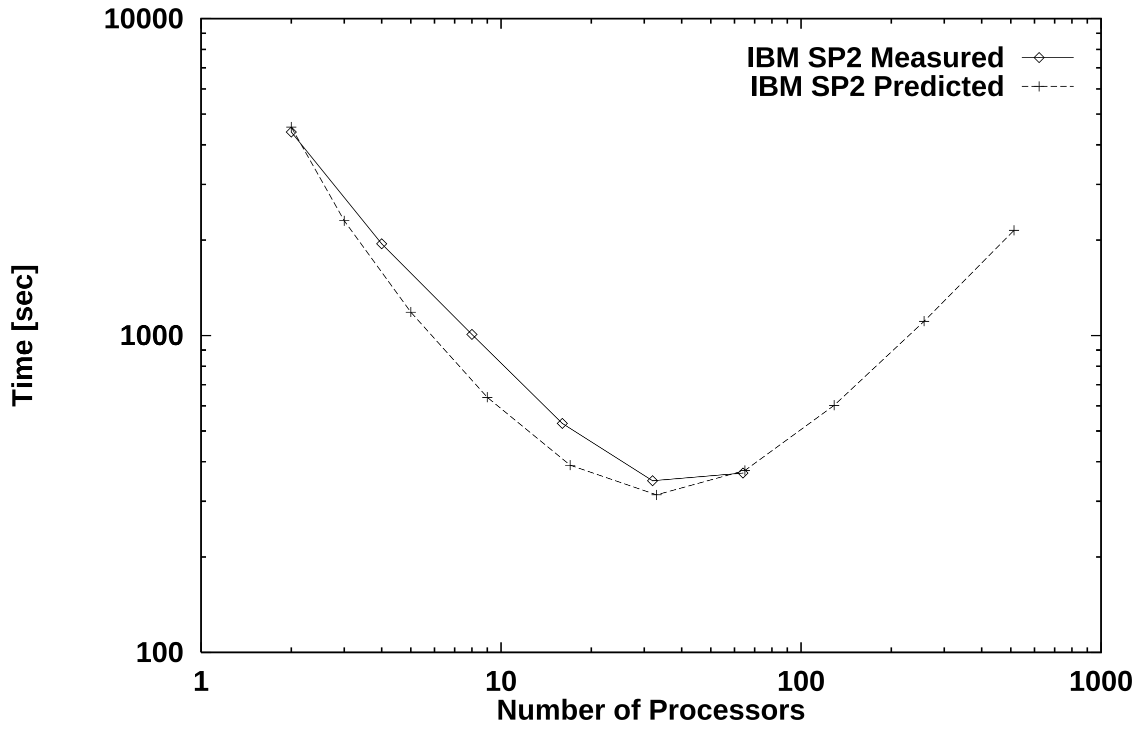

Figure 1d. Actual versus predicted wall-clock time for the CIT AQM.

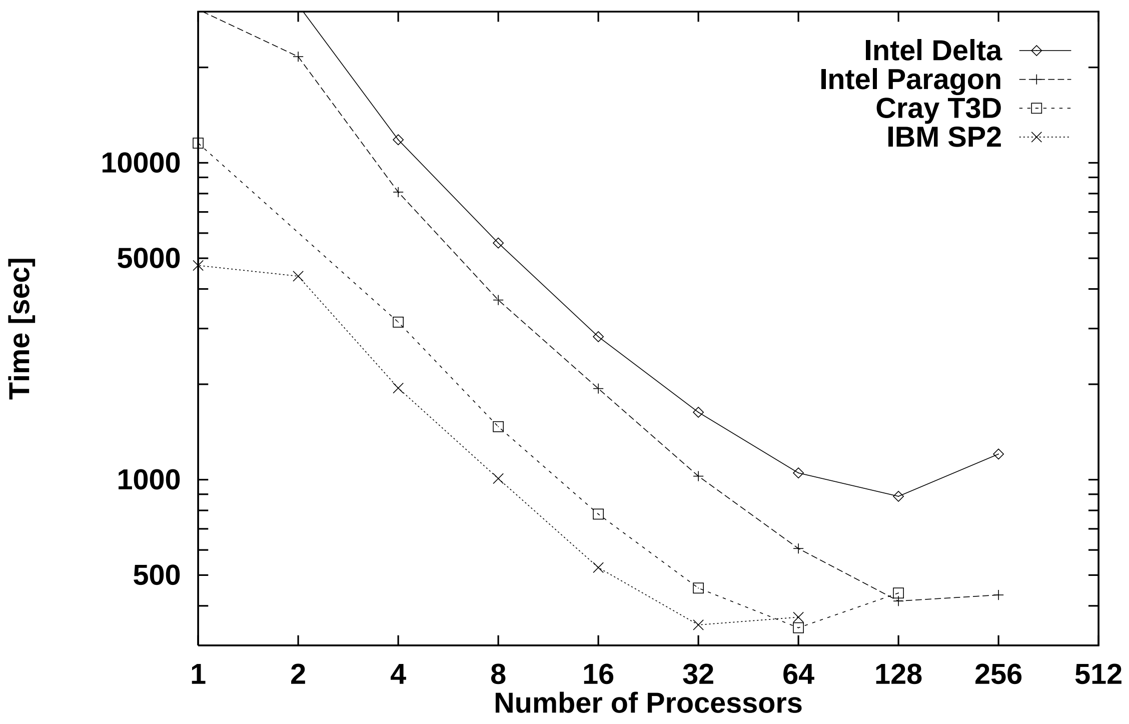

Figure 2. Performance of the parallel CIT AQM on various machines.

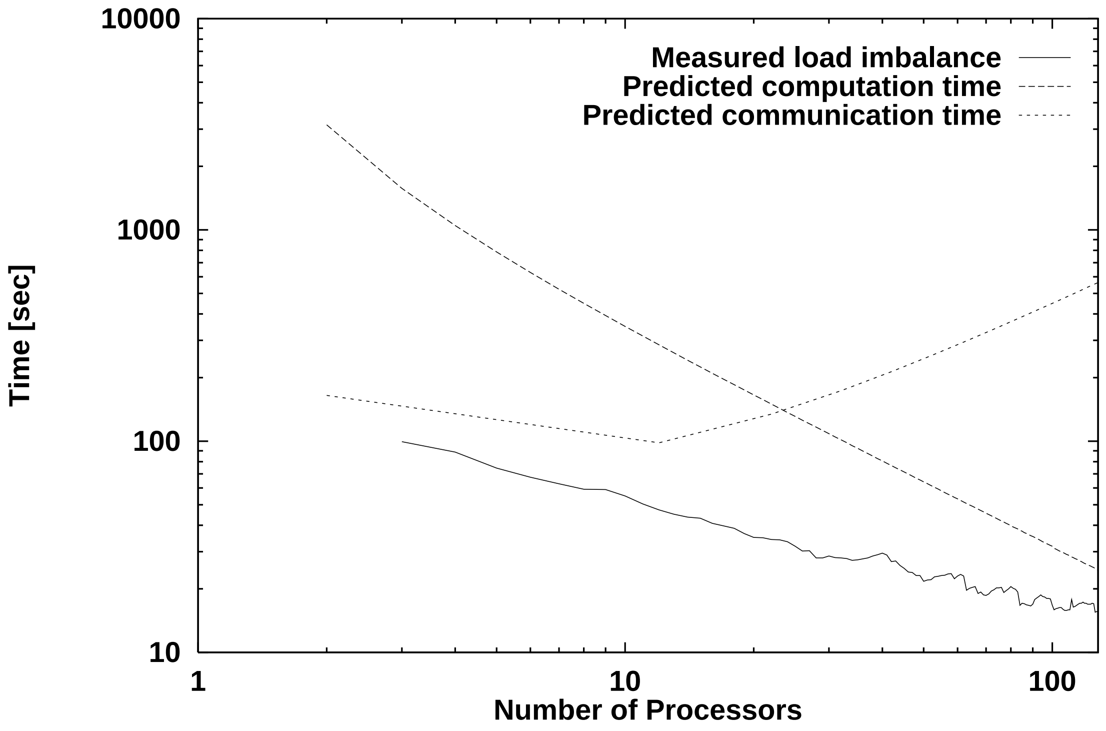

Figure 3. Measured load imbalance, predicted computation and predicted communication times on an IBM SP2.