PERGAMON

Atmospheric Environment 36 (2002) 3917–3930

ATMOSPHERIC ENVIRONMENT

www.elsevier.com/locate/atmosenv

A communication library for the parallelization of air quality models on structured grids

Philipp Miehe[a], Adrian Sandu[a], Gregory R. Carmichael[b,*], Youhua Tang[b], Dacian Dăescu[c]

[a] *Department of Computer Science, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931, USA*
[b] *Center for Global and Regional Environmental Research, The University of Iowa, Iowa City, IA 52240, USA*
[c] *Institute for Mathematics and its Applications (IMA), University of Minnesota, 400 Lind Hall, 207 Church Street S.E., Minneapolis, MN 55455-0436, USA*

Received 02 October 2001; accepted 07 May 2002

Abstract

PAQMSG is an MPI-based, Fortran 90 communication library for the parallelization of air quality models (AQMs) on structured grids. It consists of distribution, gathering and repartitioning routines for different domain decompositions implementing a master–worker strategy. The library is architecture and application independent and includes optimization strategies for different architectures. This paper presents the library from a user perspective. Results are shown from the parallelization of STEM-III on Beowulf clusters. The PAQMSG library is available on the web. The communication routines are easy to use, and should allow for an immediate parallelization of existing AQMs. PAQMSG can also be used for constructing new models. © 2002 Published by Elsevier Science Ltd.

Keywords: Air quality models; Structured grids; Parallel computing

1. Introduction

Comprehensive air quality computer models are widely used investigation tools in environmental research, in which many physical and chemical processes are modeled and their integrated impacts on atmospheric pollutant concentrations studied. Air quality models (AQMs) are also important tools for regulatory and policy communities. The clean air act stresses the importance of assessing and managing air pollution levels to protect human health and the environment. AQMs are used to develop optimal emission control strategies for atmospheric pollutants, as required by the National Ambient Air Quality Standards. Air quality models are computationally intensive applications. Incorporation of more detailed chemistry and size resolved aerosol chemistry and physics increase the estimated computational requirements by an order of magnitude. In the future AQMs will be more widely used in data assimilation and forecasting activities, which will further increase the computational requirements. In this paper we present the tools to facilitate the parallel implementation of AQMs.

2. The parallelization problem

2.1. Air quality models on structured grids

The theoretical basis for air pollution modeling is the mass balance equation (Carmichael et al., 1996):

$$\frac{\partial C_i}{\partial t} + \frac{\partial (U_j \cdot C_i)}{\partial x_j} = \frac{\partial}{\partial x_j}\left[K_{jj} \cdot \frac{\partial C_i}{\partial x_j}\right] + R_i + E_i, \quad 1 <= i <= \mathrm{Ns}. \tag{1}$$

*Corresponding author.
E-mail addresses: pmiehe@mtu.edu (P. Miehe), asandu@mtu.edu (A. Sandu), gcarmich@cgrer.uiowa.edu (G.R. Carmichael), ytang@cgrer.uiowa.edu (Y. Tang), daescu@ima.umn.edu (D. Dăescu).

1352-2310/02/$ - see front matter © 2002 Published by Elsevier Science Ltd.
PII: S1352-2310(02)00338-2

There are Ns chemical species considered; C_i stands for the concentration of species i, t for simulation time, U_j are the velocity components and x_j the spatial coordinates (three dimensional). K_{jj} are the diffusivities, R_i the rate of chemical reactions, and E_i the rate of emissions.

For a numerical solution the mass balance equation (1) is discretized on a grid that covers the part of the atmosphere of interest. Many present day AQMs, as well as meteorological simulation codes, are based on structured grids. For example MM5[1] uses nested grids to allow variable spatial resolution. Code preprocessing (with the Fortran loop and index converter FLIC, Michalakes, 1997a) and the RSL communication library (Michalakes, 1997b) have been used to achieve a flexible MM5 parallelization. The scalable modeling system (SMS) (Govett, 2000) developed at NOAA/FSL uses a directive-based approach for the parallelization of weather and climate models. New standards are currently being introduced in the weather research and forecasting (WRF) model[2] in America and new models at the European Centre for Medium-Range Weather Forecasts (ECMWF).[3] The software design principles and the two-level parallelization strategy of WRF is detaliated in Michalakes et al. (1998). Recently the use of non-structured grids in AQMs has been investigated (Tomlin et al., 1999). Such grids use nested structures based on tetrahedral or rectangular cell shapes, where selected areas of the grid can have higher resolutions than the rest of the grid. Non-structured grids will not be considered in this work. Nested structured grids are not treated explicitly in this paper, but they can also be parallelized using the data types implemented in our library.

A structured grid can be logically mapped onto a parallelipipedic lattice with $Nx \times Ny \times Nz$ points. In practice the grid is non-uniform, as it takes into account the Earth curvature and orography. More general, the grid can be set in a modified set of coordinates (e.g. latitude–longitude–pressure). The geometric shape of the modeled region can be arbitrary and has no impact on the parallelization algorithm; the only requirement is that the grid is logically structured (such that the underlying data structures are multidimensional matrices as outlined below). For simplicity we denote the three spatial dimensions by x, y, z, but the grid does not have to be aligned with the latitude–longitude system of coordinates (Table 1).

[1] Web Page. MM5 - Pennsylvania State University/ National Center for Atmospheric Research (PSU/ NCAR) mesoscale model. http://www.mmm.ucar.edu/mm5/mm5-home.html.

[2] Web Page. WRF - Weather Research and Forecasting Model. http://www.wrf-model.org.

[3] ECMWF European Centre for Medium-Range Weather Forecasts. http://www.ecmwf.int/.

2.2. Data types

The data structures for simulation on structured grids are arrays. The main array types used by AQMs are described in Table 2. The species concentrations, the major data of interest, are stored in a four dimensional array (4D) containing the concentration of each species in each grid cell. Note that for multiple phases several concentration arrays might be needed. Emission rates for each species in each grid cell are also represented by a 4D array. The wind velocity components and horizontal and vertical diffusion are one scalar per grid point, and can be represented by 3D arrays. Boundary information is held in arrays having two spatial and one species dimension. Domain top layer height is represented by a 2D array. For a complete list of the array types implemented in PAQMSG the reader should consult Miehe and Sandu (2001) and Miehe (2001).

2.3. Analysis of data dependencies

An operator split algorithm advances the solution in time using a succession of computational steps:

$$C(t^{n+1}) = T_x^{\Delta t} \cdot T_y^{\Delta t} \cdot T_z^{\Delta t} \cdot R^{2\Delta t} \cdot T_z^{\Delta t} \cdot T_y^{\Delta t} \cdot T_x^{\Delta t} \cdot C(t^n),$$

$$t^{n+1} = t^n + 2\Delta t, \tag{2}$$

where $T_x^{\Delta t}, T_y^{\Delta t}, T_z^{\Delta t}$ are the transport operators in x, y and z directions (accounting for advection by wind and turbulent diffusion), and R is the chemistry step. In more complex models R may also contain particle dynamics, chemical equilibria calculations, etc. In general R calculates the effect of local (non-spatially coupled) processes. This splitting is symmetric (cf. Strang, 1968) in the sense that different steps alternate first in a direct, then in a reverse order.

The computation of local processes R typically takes more than 90% of the total CPU time. This is an ideally parallelizable task, since the (local) computations at each grid point are independent from one another.

On the other hand the transport steps T introduce dependencies between concentrations in different grid cells. A time-explicit transport scheme requires information from the neighboring grid cells (a fixed stencil). More exactly, as illustrated in Fig. 1, the calculation of c_i^{n+1} (the concentration in grid point i at step $n+1$) requires $c_{i-k}^n, \ldots, c_{i+k}^n$ (the previous step concentrations in $2k+1$ neighboring grid points). A time-implicit transport scheme requires the whole row (or column) of grid data along the transportation direction to be present at the same processor; this is also illustrated in Fig. 1. In addition, spectral techniques used in some AQMs (Dabdub and Seinfeld, 1994a) also require an

Table 1
Interaction of the library with the model source code. Only minor changes in the serial driver (left) are required to obtain a parallel version (right). The same numerical subroutines are used by both the serial and the parallel versions of the model

Serial driver	Parallel driver
	CALL MAPPING(*Nx*, *Ny*, *Nz*,Nvloc,Nhloc)
ALLOCATE(C, U, V, W)	IF (Master) ALLOCATE(C, U, V, W)
	ELSE ALLOCATE(Ch, Cv, Uh, Vh, Wv)
CALL INPUT(C) ! Initial	IF (Master) CALL INPUT(C)
DO IT = 1,NITER	DO IT = 1,NITER
CALL INPUT(U,V,W)	IF (Master) CALL INPUT(U,V,W)
	CALL HDISTRIBUTE(U,Uh,V,Vh)
DO IT2 = 1,NITER2	DO IT2 = 1,NITER2
CALL TRANX(C,U)	CALL TRANX(Ch,Uh)
CALL TRANY(C,V)	CALL TRANY(Ch,Vh)
	CALL SHUFFLE_H2V(Ch,Cv)
CALL TRANZ(C,W)	CALL TRANZ(Cv,Wv)
CALL CHEM(C)	CALL CHEM(Cv)
CALL TRANZ(C,W)	CALL TRANZ(Cv,Wv)
	CALL SHUFFLE_V2H(Cv,Ch)
CALL TRANY(C,V)	CALL TRANY(Ch,Vh)
CALL TRANX(C,U)	CALL TRANX(Ch,Uh)
END DO	END DO
	CALL HGATHER(Ch,C)
CALL OUTPUT(C)	IF (Master) CALL OUTPUT(C)
END DO	END DO

Table 2
The main array types for a structured grid simulation. The grid has $Nx \times Ny \times Nz$ points, and there are Ns chemical species

Type	Dimension	Example
4D array	(*Nx*, *Ny*, *Nz*, *Ns*)	Concentrations; emission rates
2D array	(*Nx*, *Ny*)	Domain top and bottom layer heights
3D array	(*Nx*, *Ny*, *Nz*)	Wind velocities; diffusion coefficients
BD*z* array	(*Nx*, *Ny*, *Ns*)	Top boundary concentrations
BD*y* array	(*Nx*, *Nz*, *Ns*)	*Y* boundary concentrations
BD*x* array	(*Ny*, *Nz*, *Ns*)	*X* boundary concentrations

entire row/column of grid data to be processed simultaneously.

It is clear that a data partitioning suitable for implicit and spectral schemes will also work for explicit methods, albeit in this case some efficiency may be lost. In this paper we consider the more general implicit type partitioning. We note that a tiled domain decomposition may be used for implicit time stepping schemes using ghost regions (Govett, 2000).

2.4. Requirements and constraints

The guiding principle in designing PAQMSG is to build parallel models that behave similarly to the serial versions. The development of a model is done in serial mode while the production runs (long simulations) are done in parallel. Specifically, the serial and parallel versions have to use the same input files and produce the same output files. The parallel code has to integrate the same science modules and numerical methods as the serial code; such modules have been developed over years of work and are continuously subject to modifications.

3. Previous and related work

Carmichael et al. (1996, 1999) describe the framework and computational challenges of air quality modeling. Complex processes are incorporated into AQMs. Many phenomena are not completely understood and have to be simplified and parameterized in order to be integrated into the models. This paper presents advantages and difficulties of parallel implementations while directing attention to the differences of parallel computers (e.g. vector machines and message passing systems). Minimizing communication is the goal for the parallel version in order to achieve high performance. Symmetric splitting (2) halves the necessary communication.

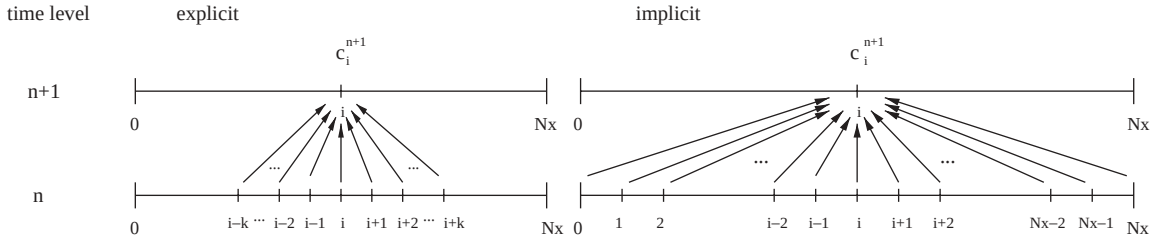

Fig. 1. Stencils for explicit and implicit transportation schemes determine different data dependencies.

Elbern (1997, 1998) put major effort into the parallelization and load balancing of the European air pollution dispersion model (EURAD). In his work he examined the contributions of dynamics, physics and chemistry modules to load imbalances and introduced a family of load balancing schemes for a wide range of platforms. Elbern used load balancing strategies based on combined global and local decision and migration bases. He compared several static and dynamic load balancing strategies using 2–128 processors on a message passing machine (Intel Paragon). The full adjustment scheme (two dimensional load balancing) produced the best results in balancing the work, achieving efficiencies from 38% (static) to 62%.

Dabdub and Seinfeld (1994b) and Dabdub and Manohar (1997) present a portable parallel implementation of an AQM for distributed memory MIMD machines. The authors developed a performance model to predict execution times of a program depending on the number of processors using machine and application-dependent parameters. This performance analysis did not count for the idle time due to load imbalances. For their experiments the authors used the California Institute of Technology (CIT) photochemical model. Their implementation used a master–worker strategy, where the master does the I/O handling and distributes the work to other processors. The communication routines are portable, being interfaced with MPI, PVM and p4. Results were obtained on different parallel machines (Intel Delta, Intel Paragon, IBM SP2, Cray T3D).

In Owczarz and Zlatev (to appear) the Danish Eulerian model (DEM) was parallelized by Owczarz and Zlatev on an IBM SMP machine. Due to the special structure of the machine general optimizations as well as machine dependent features were integrated into the code by the authors. The authors show that optimizing the serial code results in benefits for the parallel implementation. Special parallelization strategies were employed to take advantage of the machine, namely OpenMP was used to run the parallel code on the shared memory part of the IBM SMP (one node only), and MPI is used for communication between nodes. This strategy lead to impressive efficiencies.

4. The parallelization approach

The parallelization approach in this work is a master–worker strategy based on domain decomposition. This strategy targets clusters of workstations/PCs and is explained in detail in this section.

4.1. The master–worker approach

A master–worker parallelization was implemented, where the master reads the input files, distributes data to workers, gathers the results and writes them into output files. The master does not take part in any of the simulation computations. This approach is consistent with the architecture of a Beowulf cluster where a central box has access to the main storage, while nodes connected by fast Ethernet have access to local storage only.

This master–worker strategy allows for overlapping of input/output with computations. The idle master can manage a dynamic load balancing strategy. Since the master is not involved in heavy computations, the central box of the Beowulf cluster (where the master process runs) remains available for other users. Recall that compilation, for example, is done on the central box. At the same time, an idle master means that the full processing power of the system is not used. Input and output is serial. Another disadvantage is the unsymmetric communication patterns, when the master takes part in data distribution and result gathering, but during computational steps only the workers communicate among themselves.

4.2. Domain decomposition

The chosen decomposition impacts the effectiveness of a parallelization strategy. It determines load balancing, distribution time, as well as the book-keeping overhead. PAQMSG implements several data decomposition strategies, assuming a homogeneous parallel platform, where each node has the same processor speed and memory capacity; Beowulf clusters are homogeneous.

4.2.1. XY-partitioning

In the XY-partitioning all points in the grid having the same y-coordinate form an X-slice of the domain, while all points with the same x-coordinate form an Y-slice. X and Y-slices are distributed to processors in a circular fashion, assigning the first slice to the first processor, the next slice to the next processor, etc. until each processor has a slice, and then starting over with the assignment to the first processor. This partition strategy is illustrated in Fig. 2. Data distributed in X-slice format can be used for X- and Z-transport and for chemistry calculations. The Y-slice format provides the layout for Y- and Z-transport, as well as chemistry. Repartitioning data on the grid, i.e. data shuffling from X-slices to Y-slices and vice versa, is necessary during each time step in order to complete all transport computations. A disadvantage of this layout is that the number of useful workers is limited by the number of Y-slices (or X-slices) available. Usually this number is on the order of tens or a few hundred; therefore the strategy works best with few processors.

4.2.2. HV-partitioning

All points in the grid having the same Z coordinate define an H-slice ("horizontal" slice) of the grid. A V-column ("vertical" column) consists of all grid points sharing the same X and Y coordinates. Both the H-slices and the V-columns are distributed to workers in a round robin fashion. The layout can be seen in Fig. 3, which identifies H-slices in comparison to V-columns.

One step of a parallel computation with HV-partitioning is shown in Fig. 4. A shuffling from H-slices to V-columns and back has to take place during each time step. The bulk of the computations take place with data in V-columns; the number of available V-columns is $N\mathrm{x}N\mathrm{y}$ (on the order of thousands). Therefore the granularity is finer compared to XY-partitioning. This ensures scalability for larger number of processors, and provides a better load balancing. An additional advantage of this partitioning is the possibility to use genuinely two-dimensional numerical methods for the horizontal transport, which may be considerably more accurate than dimensional splitting schemes.

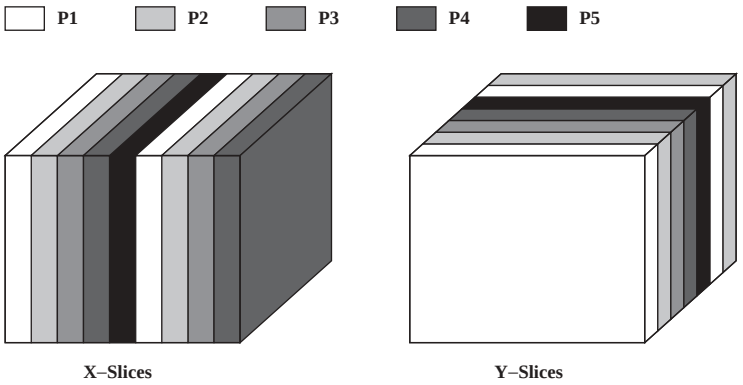

Fig. 2. Data layout for XY-partitioning. In this example there are five workers, $N\mathrm{x} = 9$, $N\mathrm{y} = 7$.

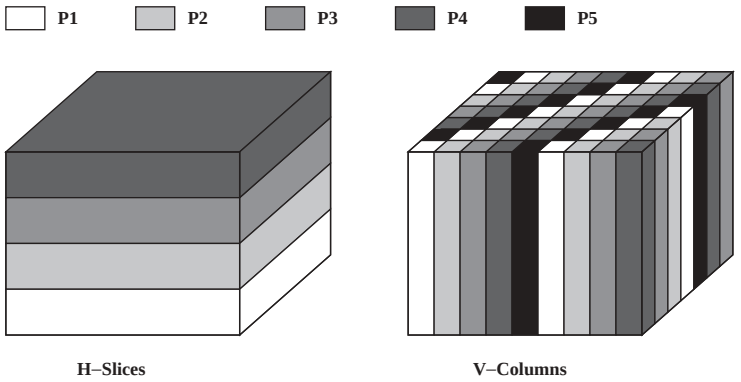

Fig. 3. Data layout for HV-partitioning. In this example there are five workers, $N\mathrm{x} = 9$, $N\mathrm{y} = 7$, $N\mathrm{z} = 4$.

Step	Who	What	Local Data Layout
1.	(Everybody)	Master reads input file and distributes data. Each worker receives data.	H-slice
2	(Each Worker)	Compute X-transport	H-slice
3	(Each Worker)	Compute Y-transport	H-slice
4	(All Workers)	Reshuffle data from H-slices to V-columns	changes
5	(Each Worker)	Compute Z-transport	V-column
6	(Each Worker)	Compute Chemistry	V-column
7	(Each Worker)	Compute Z-transport	V-column
8	(All Workers)	Reshuffle data from V-columns to H-slices	changes
9	(Each Worker)	Compute Y-transport	H-slice
10	(Each Worker)	Compute X-transport	H-slice
11	(Everybody)	Each worker sends concentration data to Master. Master writes output file.	H-slice

Fig. 4. One step of a computation in HV layout.

4.2.3. Subdomain partitioning

Although not implemented here, another partition is possible and frequently employed. Each processor can be assigned a contiguous set of grid points, i.e. a contiguous subdomain. This partitioning is suitable for explicit transportation schemes, where load balancing strategies based on dynamic adjustment of subdomain boundaries can be applied.

4.3. Communication costs

With a subdomain partitioning (and an explicit transport scheme) the data exchanged at each time step consists of concentration fields along subdomain boundaries. With XY and HV partitionings, however, a complete exchange of data between workers at each time step is necessary (the full information in a 4D concentration array is reshuffled). The communication time associated with the total exchange decreases with increasing number of processors, and remains a small percentage of the total communication time.

4.4. Load balancing

An important factor affecting the performance of the parallel code is the level of load imbalance during the execution. If the work is not evenly distributed, some processors will stay idle waiting for the others to finish, and the total computational time will be determined by the slowest processor. Several load balancing schemes have been investigated by different authors as discussed in Section 3. Traditionally, dynamic load balancing schemes work with subdomain partitions and explicit transport algorithms. Load balancing is achieved by periodically adjusting the boundaries of the subdomains.

Here we concentrate on the data decompositions XY and HV that can accommodate implicit transport schemes. A possible strategy for dynamic load balancing is to have the master keep a pool of unprocessed V-columns, and to have workers request the data, process the V-column, and return results, then request a new V-column, etc. A watch-dog can also be used to implement a fault-tolerant system (i.e. if the results are not returned within a prescribed maximal time, the V-column is made available to a different worker).

The present library implements a much simpler strategy with excellent results. In air quality modeling load imbalances appear when a region of high chemical activity is localized within a subdomain, and assigned to a single processor. The chemical solver automatically takes smaller time steps to meet the accuracy demands, and this increases the computational burden of the processor. Note that with adaptive grids load imbalances may result after grid refinement. The discussion of this situation is outside the scope of this paper.

The HV-partitioning approach offers the possibility to cover the regions of higher chemical activity with columns assigned to different processors. Thus load

imbalance is alleviated, since each worker gets a share of the higher work area. This "inherent" load balancing can be achieved if the V-columns assigned to one worker are spread uniformly over the computational domain; a clustering of V-columns of one worker in the same area has to be avoided, therefore an irregular distribution of V-columns is sought for.

Our library implements several strategies for assigning V-columns to processors. They are based on a round robin approach, where the domain is traversed by rows, columns, or diagonals, and the next V-column is assigned to the next processor. A greedy-type algorithm can be used to ensure that neighboring V-columns are not assigned to the same processor where one looks at the already assigned neighbors of the V-column at hand and then assigns this V-column to the lowest-rank processor that does hold a neighbor. Several possible strategies are illustrated in Fig. 5: diagonal round robin assignment and x–y round robin and diagonal assignments with the greedy algorithm to prevent mapping neighbors onto the same processor. One notices the (reasonably) uniform spread of V-columns over the computational domain.

5. The communication library

Efficient implementation of the communication routines has a major impact on the parallel performance (speedup and scalability). In this section we describe the general communication library PAQMSG. The routines are architecture independent but need an MPI-implementation and a FORTRAN90 compiler. The library implements distribution, shuffling and gathering routines for the array types described in Section 2.2. Application of the library assumes that the parallelization is conducted on a uniform parallel machine.

5.1. Communication routines

The master holds copies of the global 2D, 3D, BD and 4D arrays as described in Section 2.2. Workers have local (small) versions of these arrays. For each global array there are two local counterparts, each holding local data in one of the complementary formats (X-slice/Y-slice or H-slice/V-column). Data moves between master (global arrays) and workers (local arrays); data also moves between workers, being reshuffled from one type of local structure to another. We now discuss the communication routines implemented by the library.

5.1.1. Distribution

The master reads the input data from input files and updates the global data structures. The data is then distributed to all the working processors, depending on the partitioning of the model. For each of the data types four different distributions are provided: X-slice distribution, Y-slice distribution, H-slice distribution, and V-column distribution.

5.1.2. Gathering

In the end of the calculations done by the working processors the 4D concentration arrays need to be gathered by the master, which writes the output file. Gathering from four different partitionings are provided: X-slice gathering, Y-slice gathering, H-slice gathering and V-column gathering.

5.1.3. Repartitioning

In order to repartition the data during the computation routines are implemented to shuffle the data from X-slices to Y-slices (and vice versa), and to shuffle the data from H-slices to V-columns (and vice versa). The reshuffling is only needed for the 4D concentration arrays. Reshuffling is an ALLTOALL communication, as each worker needs to exchange data with every worker.

5.1.4. Implementation variants

Each of the communication routines is available in three different implementations:

- Version 1 uses one message for each slice/column to be communicated. This is the most general form and

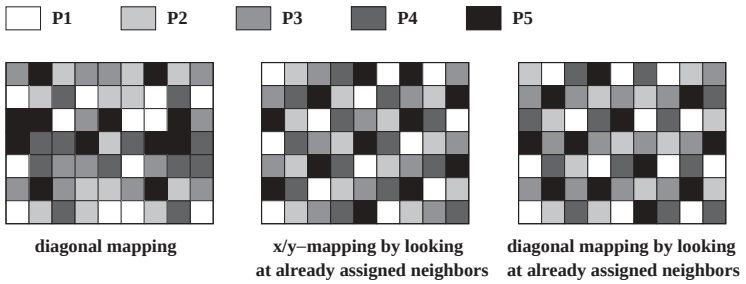

Fig. 5. Different V-column distributions. In this example $N\text{workers} = 5$, $Nx = 9$, $Ny = 7$.

is able to support irregular *HV*-mappings and the overlap of computation and communication;
- Version 2 uses one message per processor pair. This implementation builds MPI data structures using MPI_VECTOR. Time savings result since explicit copying is avoided and the number of messages is small;
- Version 3 calls the highest-level specific function provided by MPI (SCATTER for distribution, GATHER for gathering, and ALLTOALL for shuffling). These functions are defined by MPI-2 standard and are not available in all MPI implementations.

The speed of the communication routines depends on the MPI implementation as well as on the architecture used. Message passing machines have a high start-up cost (latency) and therefore give better results when fewer, longer messages are used, while shared memory machines can show good results when many short messages are sent. A setup program is provided to test and time automatically the versions available, and to recommend the fastest ones.

6. Using the library

The library is organized as a set of Fortran modules that enclose the relevant subroutines and data types. The modules are contained in several files as explained next.

6.1. The user view of the library

Module description and interrelation. The module `ParallelCommunication` includes all the other modules using the FORTRAN `use`-statement. A parallelization has to use only this module to allow all library functions to be called. Fig. 6 displays the interrelation between all modules.

The module `ParallelDataMap` describes the mapping (the layout) of data onto different processors, and initializes global variables. The module `CommDataTypes` defines the communication data structures like slices, columns and sets of slices or columns. This is only needed in the actual communication routines hosted by `CommunicationLibrary`. `ParallelMemAlloc` is a stand-alone module that contains allocation functions for the global and local data structures. These functions can be directly called by the user and are not required in any other routines.

In order to implement a new partitioning the user can either introduce a new module or change a given one while preserving the module hierarchy.

File structure. The list of library files and the distribution of modules is presented in Table 3. The main component `mpi_commlibrary.f` contains the communication routines. The definitions of the data partition scheme and the necessary global variables can be found in `mpi_util.f` and `mpi_util_mpich.f`. The files (`mpi_xy_setup.f`, `mpi_hv_setup.f`) contain the setup routines which time each communication version and help choose the best one for the architecture at hand. In addition, the library contains a set of files that are specific to the parallel implementation of STEM-III: `mpi_communication.f`, `mpi_memalloc.f`, `mpi_aq_driver_xy.f` and `mpi_aq_driver_hv.f`. The functions implemented can serve as general templates for the parallelization of AQMs.

Global variables. The global variables displayed in Table 4 help create an environment where data

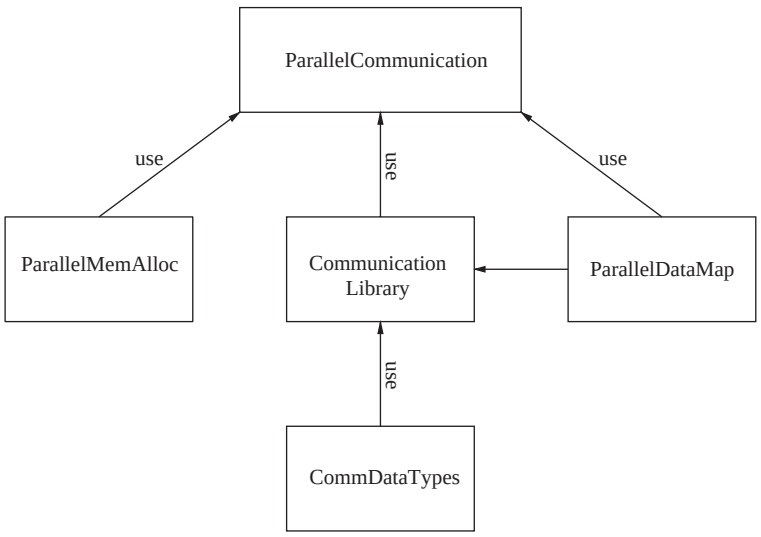

Fig. 6. Module hierarchy.

Table 3
List of library files

File	Description	Modules contained
`mpi_xy_setup.f`	Setup program for XY	
`mpi_hv_setup.f`	Setup program for HV	
`mpi_util.f`	Communication data types (MPI-2)	`{XY,HV}CommDataTypes` `{XY,HV}ParallelDataMap`
`mpi_util_mpich.f`	Communication data types (MPICH)	`{XY,HV}CommDataTypes` `{XY,HV}ParallelDataMap`
`mpi_communication.f`	STEM-III specific communication	`{XY,HV}ParallelCommunication`
`mpi_commlibrary.f`	General communication library	`{XY,HV}CommunicationLibrary`
`mpi_memalloc.f`	Memory allocation functions	`{XY,HV}ParallelMemAlloc`
`mpi_aq_driver_xy`	Parallel driver, XY-partitioning	
`mpi_aq_driver_hv`	Parallel driver, HV-partitioning	

Table 4
Global variables

Variable name	Description
Nprocs	Number of processors available
$N\{X, Y, H, V\}$ workers	Number of active workers for $\{X, Y, H, V\}$
MyId	Processor id
Master	True for master processor (processor with id = 0)
$\{X, Y, H, V\}$ Worker	True for all processors with $1 \leqslant \mathrm{id} \leqslant N\{X, Y, H, V\}$ workers

partitioning and repartitioning schemes can be applied. Global variables are declared in the `ParallelDataMap` module.

6.2. Interaction of the library with the model source code

The general structures of the serial and parallel drivers are presented in Table 1 in Fortran90-style pseudocode. Drivers manage the allocation of global and local data structures, the input and output, and call the computational routines. The presentation is simplified (e.g. the boundary conditions are omitted). The parallel version uses the same computational routines (for solving transport and chemistry) as the serial one, and in practice their interfaces are a little bit more complex then presented here. The parallelization is straightforward, as it only requires the insertion of several calls to the proper communication routines in the driver.

7. Results

To illustrate the functionality of the library we implemented a parallel version of the STEM-III AQM (Carmichael et al., 1996). In this section the results of this implementation on Beowulf clusters are reported.

7.1. STEM-III

A detailed description of STEM-III and its mathematical and scientific methods can be found in CGRER Homepage.[4] STEM-III uses structured grids. The computational cycle consists of a double loop. After setting up the grid and reading part of the input the outer loop starts. The input necessary for the computations in the current step is read; e.g. meteorological data. The inner loop performs operator-split computational steps of transport and chemistry. At the end of the outer loop the computed concentration fields are written to output files. The library allows a straightforward parallelization of STEM-III.

7.2. Beowulf clusters

Our main testing platforms were Beowulf clusters, which have recently become very popular in air pollution studies. A Beowulf system is a collection of (commodity hardware) PCs running an open-source network operating system (Linux), interconnected by a private high-speed network, and configured to operate as a single parallel computing platform. Typically the cluster is connected to the outside world through only a single node; the other nodes are not accessed as individual computers, but are dedicated to running cluster jobs. The driving design philosophy of a Beowulf system is to achieve the best possible price/performance ratio for a given computing problem.[5]

[4] Center for Global and Regional Environmental Research at University of Iowa. Homepage. http://www.cgrer.uiowa.edu.

[5] Beowulf clusters of workstations - general information. http://www.chem.arizona.edu/theochem/beowulf/whatis.html, http://www.supercomputer.org.8080/FAQ/beowulf-faq.html.

Table 5
Measured communication times (in s) for minimal and maximal number of workers

Communication	Array	X-slice	Y-slice	H-slice	V-column
Distribution	2D	0.005–0.01	0.005–0.01		0.005–0.02
	3D	0.06	0.06	0.06	0.06
	BD$\{x,y,z\}$	0.2	0.2	0.2	0.2
	4D	5–6	5.5–7	6–7	6–7
Gathering	4D	6	6	6	7–7.5
Shuffling	4D	3.5–0.5	3.5–0.5	3–0.5	3–0.5

The Beowulf cluster at Michigan Technological University[6] was used for the computational experiments. It has 64 nodes, each consisting of a 200 MHz dual-processor Pentium Pro with 128 MB memory, connected via Fast Ethernet (100 Mb/s) and a 72-port Foundry switch. Each node runs Red Hat Linux, kernel version 2.2.12. We used the MPICH[7] implementation of the MPI library.

7.3. Run times

The following formula can be used to predict parallel run times:

$$t_{\text{par}} = \frac{t_{\text{serial}} - t_{\text{io}}}{N} + t_{\text{io}} + t_{\text{distrib}}(N) + t_{\text{shuffle}}(N) + t_{\text{gather}}(N), \tag{3}$$

where t_{serial} is the serial time, t_{par} the parallel time, t_{io} the serial time spent for input/output that is not parallelizable, t_{distrib} the communication time–distribution, t_{shuffle} the communication time–shuffle, t_{gather} the communication time–gather, and N the number of workers.

The communication times depend on the architecture, the number of workers, as well as on the implementation of the communication routines (different implementations use different numbers of messages of different lengths).

For data distribution/gathering the master sends/receives data to/from all workers. The implementation could invoke a number of messages proportional to the number of workers; the length of each message decreases as the number of workers increases. No overlapping of these messages is possible as the master is involved in every communication. In this implementation the total time for distributing m words of data can be approximated by the idealized model

$$t_{\text{distribution}} = N\left(t_{\text{startup}} + t_{\text{word}}\frac{m}{N}\right) = N\,t_{\text{startup}} + t_{\text{word}}\,m.$$

The distribution time is composed of a constant transfer time $t_{\text{word}}\,m$ plus a linearly increasing startup time component. This conclusion agrees well with the measurements presented in Table 5, which show a slight increase in distribution and gathering times for an increasing number of workers. If one uses an implementation based on one-to-all operations (scatter/gather), and these operations use a tree-based communication pattern, there are $\log_2 N$ communication stages and in each stage j one transfers $m/2^j$ words of data in point to point operations. A simple model for the total distribution time is then

$$t_{\text{distribution}} = \sum_{j=1}^{\log_2 N}\left(t_{\text{startup}} + t_{\text{word}}\frac{m}{2^j}\right) = \log_2 N\,t_{\text{startup}} + t_{\text{word}}\,m\left(1 - \frac{1}{N}\right).$$

Again the communication time has a fixed large component $t_{\text{word}}m$ plus a slowly increasing component, and this model prediction is also in line with Table 5 measurements.

Shuffling between the workers on the other hand has no single point of connection, and message overlapping is possible. The amount of data sent in a shuffling procedure is the same as in the distribution of a 4D array, but split into much smaller messages. An idealized model for all-to-all communication time for m words of data is

$$t_{\text{all}}\text{-to-all} = \log_2 N\left(t_{\text{startup}} + t_{\text{word}}\frac{m}{N}\right).$$

For a moderate number of workers and large messages the first term $t_{\text{startup}}\log_2 N$ is small compared to the second, and the shuffling time decreases with the number of workers at the approximate rate $(\log_2 N)/N$. This simple model predicts that the shuffling time for 64 workers is about 5.3 times smaller than the shuffling

[6] Echtheow - The Beowulf cluster at Michigan Technological University. http://www.cs.mtu.edu/beowulf.

[7] Mathematics and Computer Science Division, Argonne National Laboratory. MPICH - A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich/.

time with two workers; the measurements shown in Table 5 (last row) show this ratio to be 1:7.

Measured communication times on the MTU Beowulf cluster are shown in Table 5. The shuffling times are reported for both the minimum ($N = 2$) and the maximum ($N = 64$) number of workers. Different times in the prediction formula (3) were also measured for 1 h of simulation time and reported in Table 6. Note that the serial computation has a simulation time (wall-clock time) to CPU time ratio less than 1:1. Input/output accounts for less than 0.6% of the serial time.

The above estimates work for a moderate number of workers ($N \sim$ low hundreds). For massively parallel architectures ($N \sim$ thousands) the startup components become very significant, which may prevent a good scaling of the PAQMSG communication times. Most practitioners however will run their parallel air quality models on medium scale platforms, and will find PAQMSG scaling to be very satisfactory.

7.4. Parallel code performance

The simulation times with different numbers of processors are presented in Table 7. The total CPU time (per 1 h simulation time) decreases from 1 h 15 min (serial) down to less than 2 min (on 64 workers).

Table 6
Serial computation time and input–output and communication overheads for parallel STEM-III, 1 h simulation time

Serial time	4480 s	
Input/output time	26 s	
Communication	XY time (s)	HV time (s)
Distribution (4D array)	6	6
Distribution (all other)	6–9	8–10
Gathering	6	6
Shuffling	21–3	19.5–3.6

Table 7
Simulation times (in seconds) of the parallel STEM-III on the MTU Beowulf cluster

Workers	XY-partitioning	HV-partitioning
2	2120.55	2070.67
4	1097.47	1061.96
8	573.77	554.64
16	333.72	327.40
32	185.50	183.11
61	119.00	122.41
64		119.44

Fig. 7 presents the percentages of the total CPU time claimed by different communication and computation steps. The computation with data in V-column format (including chemistry and Z-transport) accounts for most of the computational effort (82% with 16 processors and 63% with 60 processors). The complete parallelization of this part explains the computational savings reported. The H computation (horizontal transport) accounts for only 3–4% of the total CPU time; the number of processors that can contribute to the H computation is bounded by the number of available H-slices (here 20).

The computational part dominates when a small number of processors is used. As more processors are employed, the computational time decreases, while the communication time remains about the same; therefore the communication time becomes increasingly important for larger numbers of processors. For example, the distribution time is constant and accounts for 3% of the time with 16 workers and for 14% of the total time with 60 workers. The total shuffling time percentage increases only modestly (i.e. shuffling scales well with the number of workers).

The efficiency and speedup diagrams are given in Fig. 8. The parallel code employing 64 workers is more than 35 times faster than the serial code. The efficiency of the parallelization remains above 50% up to 64 workers for the HV-scheme. The efficiency drops of the XY scheme are due to the load imbalances, as explained next.

7.5. Load imbalance

Load imbalance has a major impact on parallel performance. The amount of load imbalance is visualized in Fig. 9. For the HV partitioning the maximal idle time, and the minimal computational time on V-columns (Z-transport and chemistry) are given. The minima correspond to the processor that gets the least amount of work. The idle time is 2% with 16 workers and 6% with 60 workers. For the XY partitioning the maximal idle time, and the minimal computational time on Y-slices (including chemistry) are given. The maximal idle time is significant, 26% with 16 workers and 48% with 60 workers. This is explained by the fact that in this example there are 61 Y-slices; 59 workers get only one Y-slice, while the remaining worker gets two. The total computational time is given by this last worker. The remaining processors compute one Y-slice, then are idle for the time needed by the processing of the second Y-slice. This is shown as a 48% idle time in the figure. We now can explain the stair-like shape of the speed-up diagram for the XY partitioning shown in Fig. 8. Whenever the number of Y-slices divides the number of workers one has a balanced workload. However, if the number of workers is between two consecutive divisors the maximal number of slices per processor is

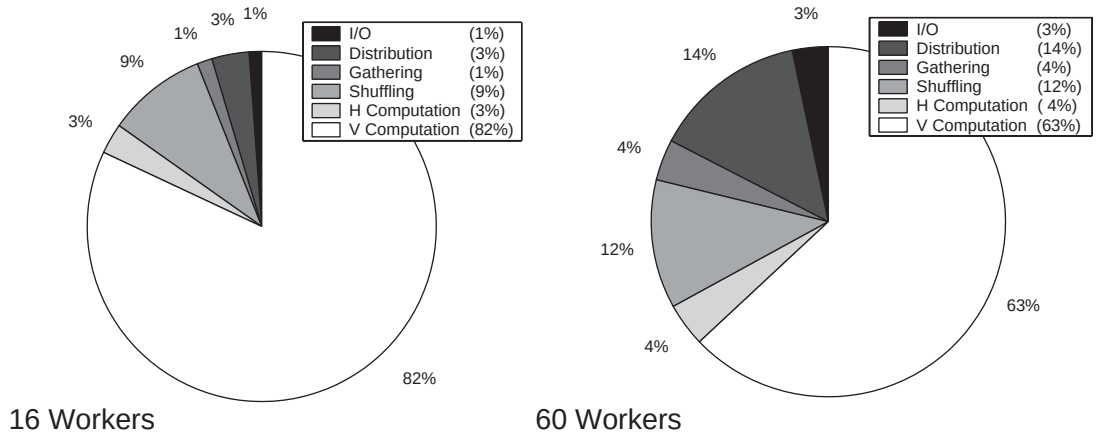

Fig. 7. Percentage of the total CPU time claimed by communication and computation in a 12 hr STEM-III simulation with HV domain decomposition.

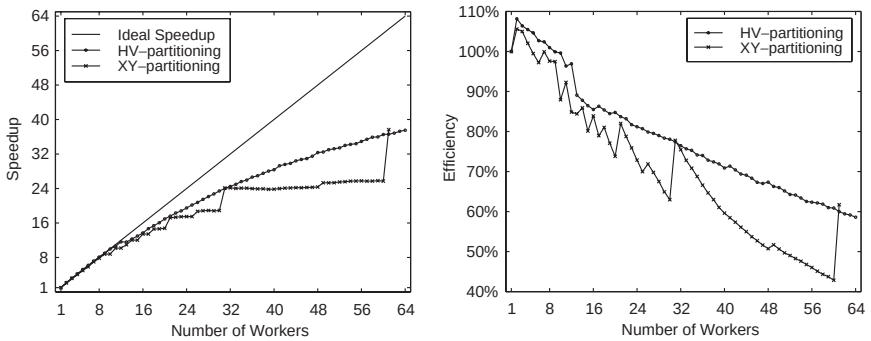

Fig. 8. Speedups and efficiencies for a 1 h STEM-III simulation with HV and XY domain decompositions.

the same, and the speed-up curve is flat (e.g. dividing 61 Y-slices to any number of workers in between 31 and 60 assigns two slices to at least one processor).

Fig. 9 confirms our expectation that the HV static load balancing scheme yields close to optimal results. The maximal load imbalance is below 6% when up to 64 workers are employed. A comparison of this approach with a dynamic load balancing scheme is left to future work.

8. Conclusions and future work

We developed a communication library for the parallelization of AQMs on structured grids. It implements a master–worker strategy and XY and HV domain decompositions. The use of the MPI message-passing layer for communication ensures portability.

Application of the library to STEM-III provides a parallelization example. The computational time is reduced from 1 h and 15 min per hour of simulation down to less than 2 min using 64 workers. The efficiency of the system is above 50%.

The HV-partitioning allows good scaling for large numbers of processors. The inherent static load balancing strategy performs remarkably well, keeping the load imbalance in the range of several percent. We believe that this approach can compete with a dynamic load balancing approach on uniform parallel machines, since its overhead is smaller.

Of course, dynamic load balancing can make the parallelization applicable to non-uniform networks of workstations, increasing the portability of the communication library. Future work will be devoted to developing and implementing various dynamic load balancing strategies.

In order to reduce the communication overhead parallel input and output routines can be employed. This has the potential to significantly reduce the overhead as the distribution and gathering steps become

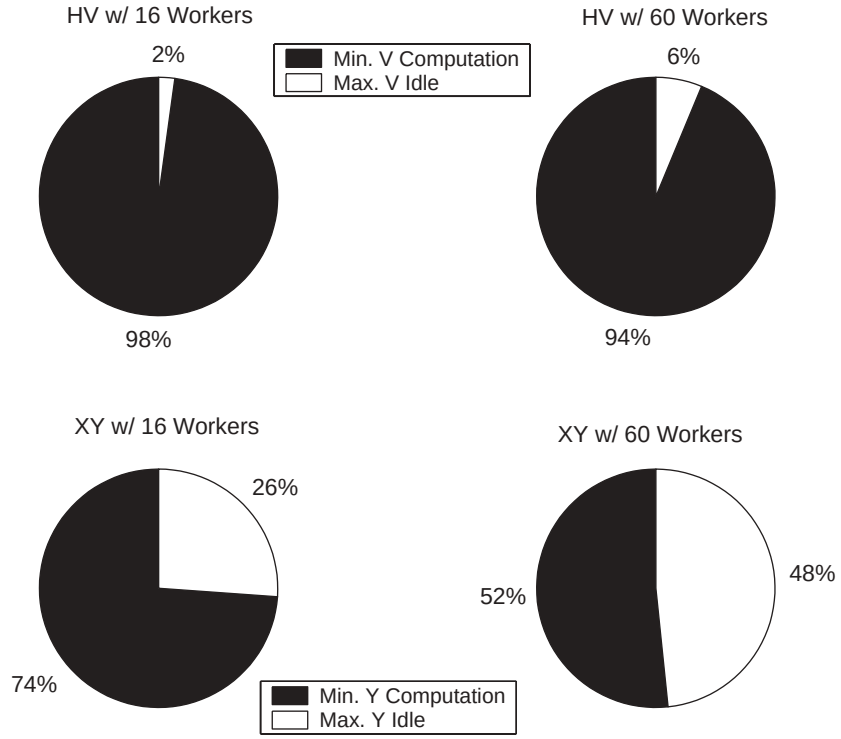

Fig. 9. Load imbalances for a 12 hs STEM-III simulation with HV and XY partitions.

unnecessary. On a Beowulf cluster, where only the central box has access to the main storage, some collision of simultaneous IO is to be expected.

Acknowledgements

The work of P. Miehe and A. Sandu was supported by the NSF CAREER award ACI-0093139 and by Michigan Technological University. Part of this work was completed by P. Miehe as a Project for the M.S. degree in the Computer Science Department at Michigan Technological University. The authors wish to thank the anonymous reviewers for their many constructive suggestions.

References

Carmichael, G.R., Sandu, A., Potra, F.A., Damian-Iordache, V., Damian-Iordache, M., 1996. The current state and the future directions in air quality modeling. SAMS 25, 75–105.

Carmichael, G.R., Sandu, A., Song, C.H., He, S., Phandis, M.J., Daescu, D., Damian-Iordache, V., Potra, F.A., 1999. Computational Challenges of Modeling Interactions Between Aerosol and Gas Phase Processes in Large Scale Air Pollution Models. Kluwer Publishers Dordrecht pp. 99–136.

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

Dabdub, D., Seinfeld, J.H., 1994a. Numerical advective schemes used in air quality models—sequential and parallel implementation. Atmospheric Environment 28, 3369–3385.

Dabdub, D., Seinfeld, J.H., 1994b. Air quality modeling on massively parallel computers. Atmospheric Environment 28, 1679–1687.

Elbern, H., 1997. Parallelization and load balancing of a comprehensive atmospheric chemistry transport model. Atmospheric Environment 31, 3561–3574.

Elbern, H., 1998. On the load balancing problem of comprehensive air quality models. Journal Systems Analysis-Modelling-Simulation 32, 31–56.

Govett, M., 2000. The scalable modeling system SMS: a high level alternative to MPI. Ninth Workshop on the Use of High Performance Computing in Meteorology. Developments in Teracomputing. European Centre for Medium-Range Weather Forecasts Shinfield Park, Reading, 13–17 November 2000.

Michalakes, J., 1997a. FLIC: a translator for same-source parallel implementation of regular grid applications. Technical Report ANL/MCS-TM-223, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

Michalakes, J., 1997b. RSL: A parallel runtime system library for regional atmospheric models with nesting. In: Baden, S., Chrisochoides, N., Gannon, D., Norman, M. (Eds.), Proceedings of the IMA Workshop on Structured Adaptive Mesh Refinement Grid Methods, Minneapolis, March 12–13, 1997. Springer, Berlin.

Michalakes, J., Dudhia, J., Gill, D., Klemp, J., Skamarock, W., 1998. Design of a next-generation regional weather research and forecast model. ECMWF Workshop, Reading, UK, November 1998. (http://www-unix.mcs.anl.gov/michalak/ecmwf98/html).

Miehe, P., 2001. Parallelization of air quality models on structured grids using MPI. MS Project Report, Computer Science Department, Michigan Technological University.

Miehe, P., Sandu, A., 2001. PAQMSG-User's Guide. A library for the parallelization of air quality models on structured grids. http://www.cs.mtu.edu/asandu/Software/Parallel/Home.html.

Owczarz, W., Zlatev, Z. Running a large-scale air pollution model on fast supercomputers. International Journal on Computer Research, to appear.

Strang, G., 1968. On the construction and comparison of difference schemes. SIAM Journal on Numerical Analysis 5, 506–517.

Tomlin, A.S., Ghorai, S., Hart, G., Berzins, M., 1999. 3-D adaptive unstructured meshes in air pollution modeling. Environmental Management and Health 10 (4), 267–274.