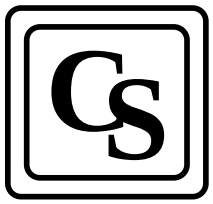

COMPUTER SCIENCE TECHNICAL REPORT

A Communication Library for the Parallelization of Air Quality Models on Structured Grids

P. Miehe, A. Sandu
G.R. Carmichael, Y. Tang, D. Dăescu

CSTR-01-05
September 2001

Michigan Technological University, Houghton, MI 49931

A Communication Library for the Parallelization of Air Quality Models on Structured Grids

September 23, 2001

Philipp Miehe, Adrian Sandu

Department of Computer Science, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931. E-mail {pmiehe,asandu}@mtu.edu.

Gregory R. Carmichael[a], Youhua Tang

Center for Global and Regional Environmental Research, The University of Iowa, Iowa City, IA 52240. E-mail {gcarmich,ytang}@cgrer.uiowa.edu.

Dacian Dăescu

Institute for Mathematics and its Applications (IMA), University of Minnesota, 400 Lind Hall, 207 Church Street S.E., Minneapolis, MN 55455-0436. E-mail daescu@ima.umn.edu.

Abstract: PAQMSG is an MPI-based, Fortran 90 communication library for the parallelization of Air Quality Models 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 air quality models. PAQMSG can also be used for constructing new models.

Keywords: Air quality models, structured grids, parallel computing.

---

[a]Corresponding Author

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 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. Air quality models 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. A basic description of the photochemical oxydant cycle requires the treatment of the chemistry, transport and deposition of about 70 gas-phase species involved in 200 coupled nonlinear chemical reactions. On today's workstations (computational power $\sim$ 1 GigaFlop/s) a discretization with $1.6 \cdot 10^5$ gridpoints (e.g. $100 \times 80 \times 20$ points) is at best solved at a 1:1 simulation time to CPU time ratio. For a regional scale application this would cover a 5000 km $\times$ 4000 km domain with a grid spacing of 50 km. A global model at this resolution requires about $4 \cdot 10^6$ gridpoints, and therefore 25 GigaFlop/s are needed for a simulation time to CPU time ratio of 1:1, and 0.25 TeraFlop/s for the desired ratio of 100:1. Incorporation of more detailed chemistry and of aerosol simulations increase the estimated computational requirements to tens of TeraFlop/s. These computational needs make parallel computing an ideal environment for Air Quality Modeling.

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. [4]):

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

Here C_i stands for the concentration of chemical species i, 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. At Pennsylvania State University the National Center for Atmospheric Research developed the mesoscale model in the early 70's, representing the standard of weather forecast models up to today's fifth generation (MM5). Now a new standard will be introduced, the Weather Model for Forecasting (WMF) in America and the European Community Weather Model for Forecasting (ECWMF) in Europe, replacing MM5. Recently the use of non-structured grids in AQMs has been investigated [13]. Such grids use nested structures based on tetrahedra or rectangular cell shapes, where selected areas of the grid can have higher resolutions then 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 $N\mathrm{x} \times N\mathrm{y} \times N\mathrm{z}$ 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). This has no impact on the parallelization algorithm; for simplicity we denote the three spatial dimensions by $\mathrm{x}, \mathrm{y}, \mathrm{z}$.

Type	Dimension	Description
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
BDz array	(Nx,Ny,Ns)	Top boundary concentrations
BDy array	(Nx,Nz,Ns)	Y boundary concentrations
BDx array	(Ny,Nz,Ns)	X boundary concentrations

Table 1: The main array types for a structured grid simulation. The grid has $N\mathrm{x} \times N\mathrm{y} \times N\mathrm{z}$ points, and there are Ns chemical species.

2.2 Data Types

The data structures for simulation on structured grids are arrays. The main array types used by an Air Quality Model are described in Table 1. 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 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 [9, 10].

2.3 Analysis of Data Dependencies

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

$$C(t^{n+1}) = T_{\mathrm{x}}^{\Delta t} \cdot T_{\mathrm{y}}^{\Delta t} \cdot T_{\mathrm{z}}^{\Delta t} \cdot R^{2\Delta t} \cdot T_{\mathrm{z}}^{\Delta t} \cdot T_{\mathrm{y}}^{\Delta t} \cdot T_{\mathrm{x}}^{\Delta t} \cdot C(t^n) \ , \quad t^{n+1} = t^n + 2\Delta t \ , \tag{2}$$

where $T_{\mathrm{x}}^{\Delta t}, T_{\mathrm{y}}^{\Delta t}, T_{\mathrm{z}}^{\Delta t}$ 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 (Strang), in the sense that different steps alternate first in a direct, then in a reverse order.

The computation of local processes R takes more than 90% of the total CPU time. This is an embarrassingly parallel 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 Figure 1, the calculation of c_i^{n+1} (the concentration in grid point i at step $n+1$) requires $c_{i-k}^n, \cdots, c_{i+k}^n$ (the previous step concentrations in $2k+1$ neighboring gridpoints). 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 Figure 1.

It is clear that a data partitioning suitable for implicit 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.

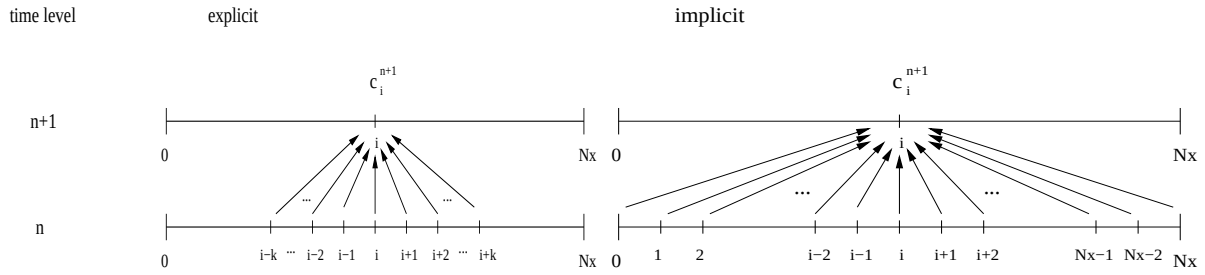

Figure 1: Stencils for explicit and implicit transportation schemes determine different data dependencies.

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 Related Work

Carmichael et. al. [4, 5] describe the framework and computational challenges of air quality modeling. Complex processes are incorporated into AQMs; most 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.

Elbern [7, 8] 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 therefore 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 to 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; specifically, it improved the efficiency from 38 percent (static) to 62 percent.

Dabdub et. al. [6] present a portable parallel implementation of an AQM for distributed memory MIMD machines. The authors developed a performance model to be able to predict execution times of a program depending on the number of processors using machine and application-dependent parameters. This performance analysis does 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 [12] the Danish Eulerian Model (DEM) is parallelized by W. Owczarz and Z. Zlatev on an IBM SMP machine. Due to the special structure of the machine not only general optimizations, but also 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 are employed to take advantage of the machine, namely openMP is 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 leads to impressive efficiency.

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 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 could manage a dynamic load balancing strategy. Since the master in 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.

In 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 pattern - the master takes part in data distribution and result gathering, but during computational steps only the workers communicate between themselves.

4.2 Domain Decomposition

The chosen partition of the work impacts the effectiveness of a parallelization strategy. It determines load balancing, distribution time as well as the amount of book-keeping effort. PAQMSG implements several data decomposition strategies, discussed further. They assume 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 at the first processor. This partition strategy is illustrated in Figure 2.

Data distributed in X-slice format could 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.

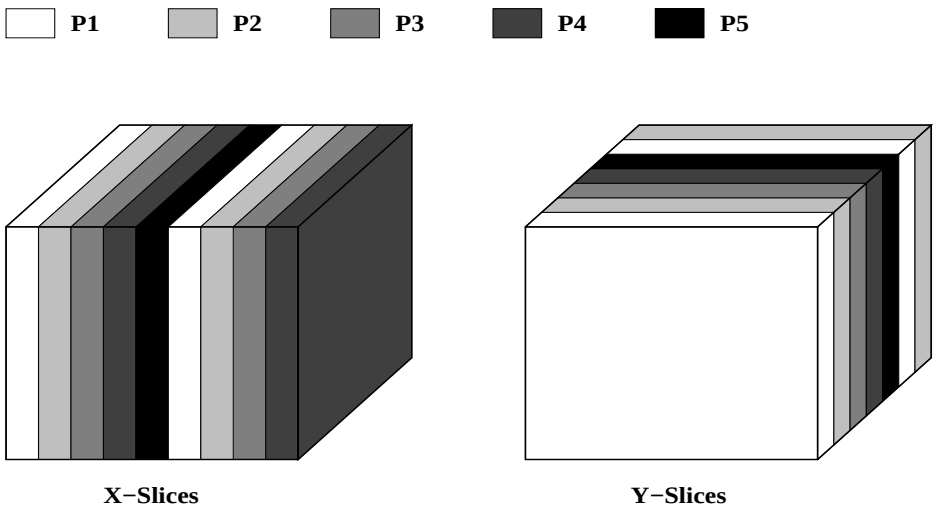

Figure 2: Data layout for XY-partitioning. In this example there are 5 workers, $N\mathrm{x} = 9$, $N\mathrm{y} = 7$.

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 Figure 3, which identifies H-slices in comparison to V-columns.

One step of a parallel computation with HV-partitioned is shown in Figure 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 $N\mathrm{x} \cdot N\mathrm{y}$ (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.

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 is necessary; the full information in a 4D concentration array is

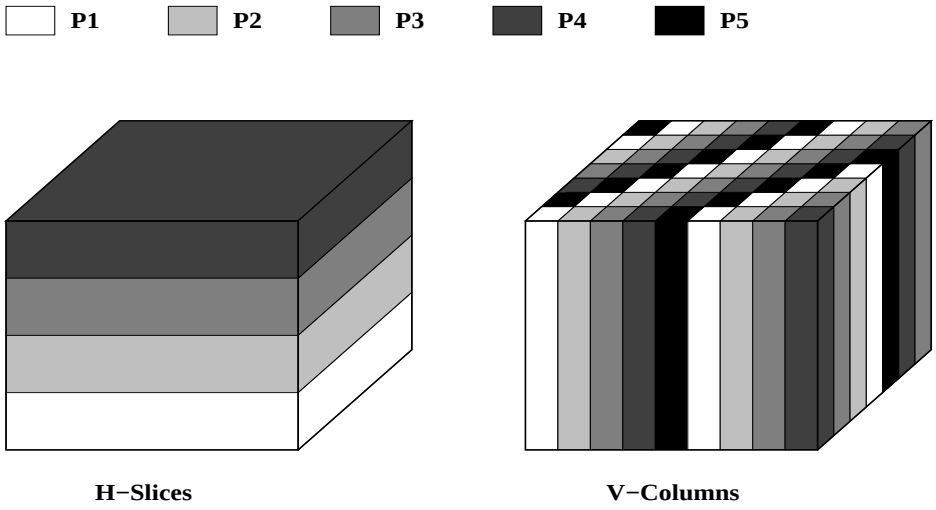

Figure 3: Data layout for HV-partitioning. In this example there are 5 workers, $Nx = 9$, $Ny = 7$, $Nz = 4$.

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

Figure 4: One step of a computation in HV layout.

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; the total computational time is then determined by the slowest processor.

Several load balancing schemes have been investigated by different authors as seen 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; 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; if the results are not returned within a prescribed maximal time, the V-column is again 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; a 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 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.

The 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. A greedy-type algorithm can be used to ensure that neighboring V-columns are not assigned to the same processor. Several strategies are illustrated in Figure 5; 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

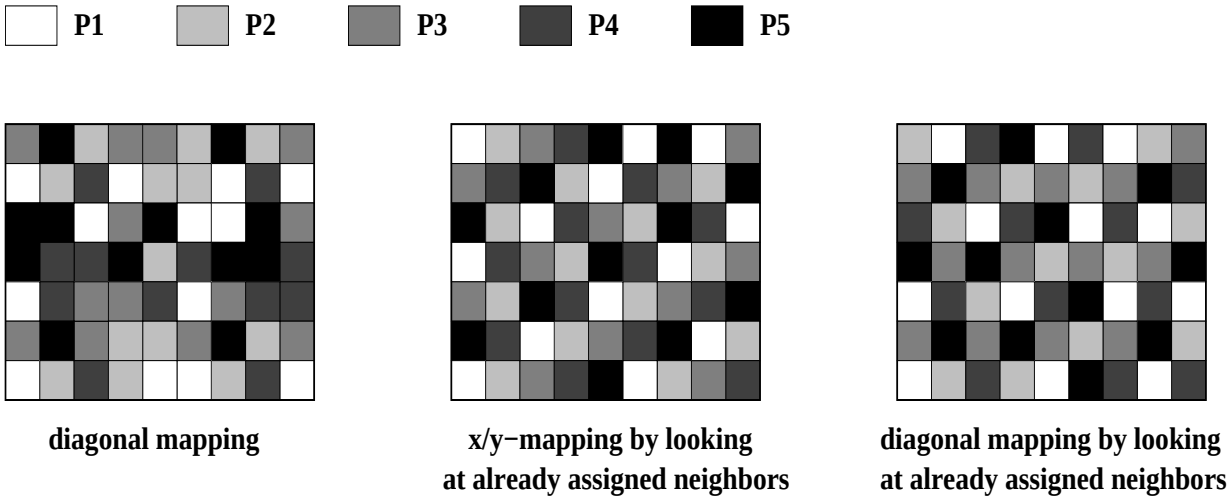

Figure 5: Different V-column distributions. In this example $Nworkers = 5$, $Nx = 9$, $Ny = 7$.

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, 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, 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; 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 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. Figure 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.

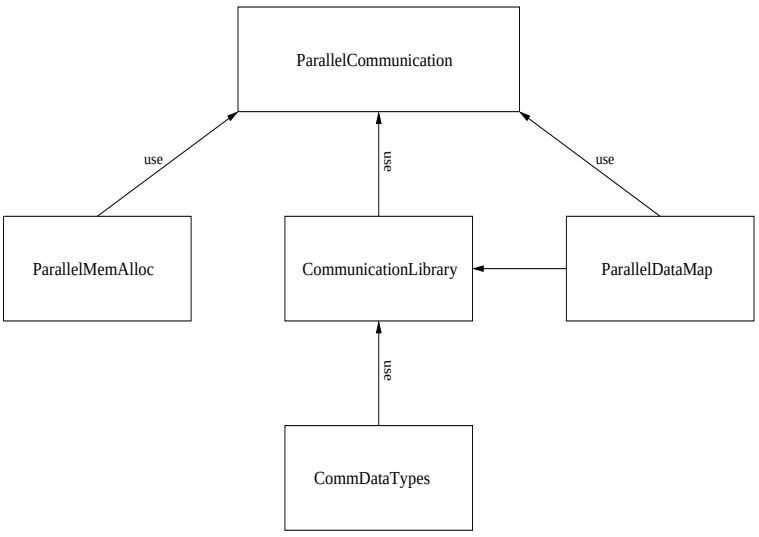

Figure 6: Module hierarchy.

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	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 2: List of library files.

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 $\leq$id $\leq$ N{X,Y,H,V}workers

Table 3: Global variables.

6.2 File Structure

The list of library files and the distribution of modules is presented in Table 2.

The main component of the library is `mpi_commlibrary.f` which contains the main communication routines. The definition of the data partition scheme and definitions of necessary global variables can be found in `mpi_util.f` and `mpi_util_mpich.f`.

The setup files (`mpi_xy_setup.f, mpi_hv_setup.f`) do not add to the library functionality, but they are necessary to choose the best implementation version for the architecture at hand. The setup routines time each communication version and write the result into an output file called `XY_versions` and `HV_versions`. One of these files is read in during initialization of the associated partition where the communication versions are chosen.

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.

A makefile is also included in order to compile the STEM-III code. Different make options, code as well as the parallel extensions have to be installed are described in the README and INSTALL files.

6.3 Global Variables

The global variables displayed in Table 3 help create an environment where data partitioning and repartitioning schemes can be applied. Global variables are declared in the `ParallelDataMap` module.

7 Results

To illustrate the functionality of the library we implemented a parallel version of the state-of-the-science air quality model STEM-III (Carmichael et. al. [4]). 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 [11]. 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 where Beowulf clusters, which recently became 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, they 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 [1].

The Beowulf cluster at Michigan Technological University [2] 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 (100Mb/sec) and a 72-port Foundry switch. Each node runs Red Hat Linux, kernel version 2.2.12.

7.3 Run Times

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

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

t_{serial}	- serial time
t_{par}	- parallel time
t_{io}	- serial time spent for input/output that is not parallelizable
t_{distrib}	- communication time - distribution
t_{shuffle}	- communication time - shuffle
t_{gather}	- communication time - gather
N	- 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) a number of messages proportional to the number of workers; the length of each message decreases as the number of workers increases. No overlapping

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

Table 4: Measured Communication Times (in seconds) for minimal and maximal number of workers.

of these messages is possible as the master is involved in every communication. 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. The shuffling time decreases with the number of workers.

Measured communication times on the MTU Beowulf cluster are shown in Table 4. The shuffling times are reported for both the minimum and the maximum number of workers. Different times in the prediction formula (3) were also measured for one hour of simulation time and reported in Table 5. 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 percent of the serial time.

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

Table 5: Serial computation time and input-output and communication overheads for parallel STEM-III, 1 hour simulation time.

7.4 Parallel Code Performance

The simulation times with different numbers of processors are presented in Table 6. The total CPU time (per 1 hour simulation time) decreases from 1 hour 15 minutes (serial) down to less than 2 minutes (on 64 workers).

Figure 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,

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

Table 6: Simulation times (in seconds) of the parallel STEM-III on the MTU Beowulf cluster.

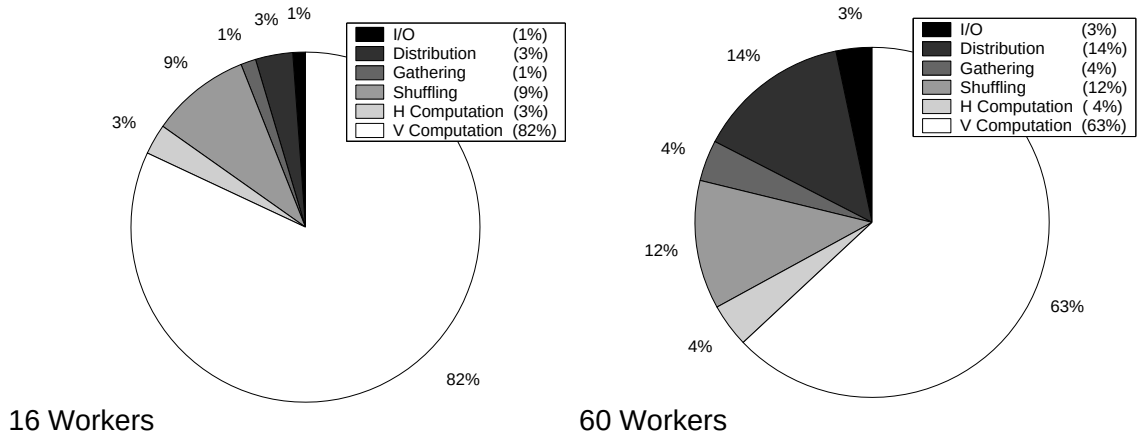

Figure 7: Percentage of the total CPU time claimed by communication and computation in a 12 hours STEM-III simulation with HV domain decomposition.

the distribution time is constant, it accounts for 3% of the time with 16 workers and for 14% of the total time with 60 workers. Note that 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 Figure 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 percent 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 Figure 9. For the HV partitioning the maximal idle time, and the minimal computational time on V-columns (Z-transport and chemistry) are given; they 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 other 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

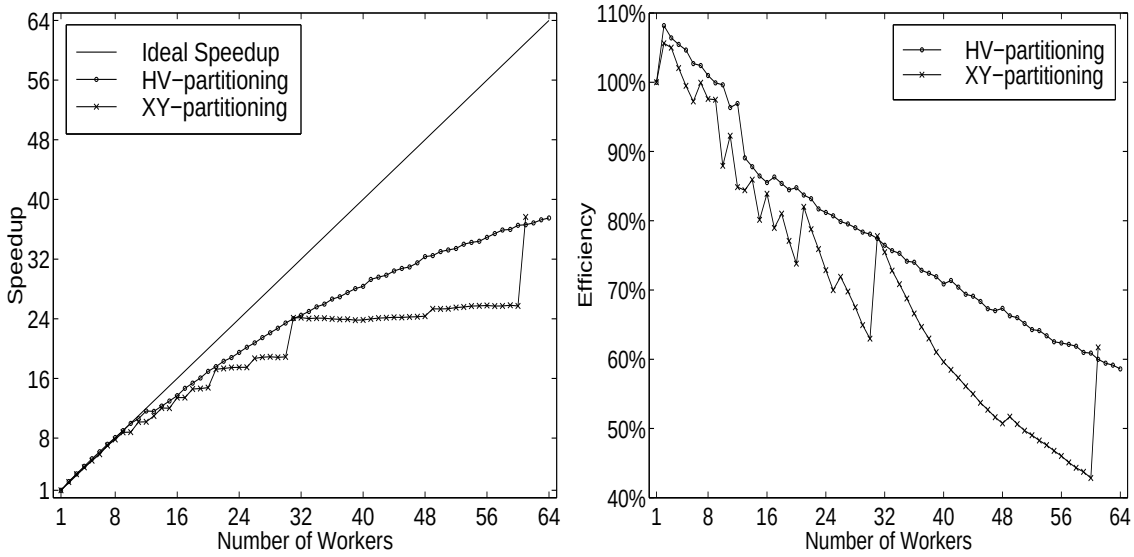

Figure 8: Speedups and efficiencies for Parallel STEM-III with HV and XY domain decompositions.

now can explain the stair-like shape of the speed-up diagram for the XY partitioning shown in Figure 8. Whenever the number of Y-slices divides the number of workers one has a balanced workload; if the number of workers is between two consecutive divisors the maximal number of slices per processor is 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 2 slices to at least one processor).

Figure 9 data 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 air quality models 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 hour and 15 minutes per hour of simulation down to less than 2 minutes 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 unnecessary. On a Beowulf cluster, where only the central box has access to the main storage, some collision

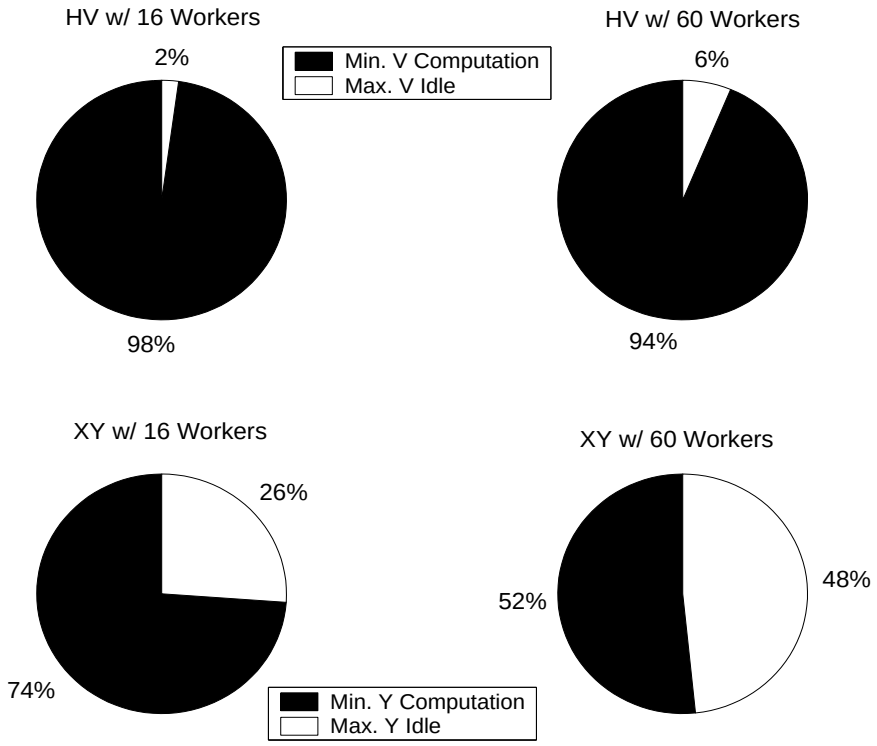

Figure 9: Load imbalances for a 12 hours STEM-III simulation with HV and XY partitions.

of simultaneous IO is to be expected.

Library Availability

PAQMSG and the user manual are available for download from
`http://www.cs.mtu.edu/~asandu/Software/Parallel/Home.html`.

Acknowledgments

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.

References

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

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

[3] G. R. Carmichael. Private Communication.

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

[5] G. R. Carmichael, A. Sandu, C. H. Song, S. HE, M. J. Phandis, D. Daescu, V. Damian-Iordache, and F. A. Potra. *Computational challenges of modeling interactions between aerosol and gas phase processes in large scale air pollution models*, pages 99–136. Kluwer Publishers, 1999.

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

[7] H. Elbern. Parallelization and load balancing of a comprehensive atmospheric chemistry transport model. *Atm. Env.*, 31:3561–3574, 1997.

[8] H. Elbern. On the load balancing problem of comprehensive air quality models. *SAMS*, 32:31–56, 1998.

[9] P. Miehe and A. Sandu. PAQMSG- User's Guide. A library for the parallelization of Air Quality Models on Structured Grids

[10] P. Miehe. Parallelization of Air Quality Models on Structured Grids Using MPI. M.S. Project Report, *Computer Science Department*, Michigan Technological University, August 2001.

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

[12] W. Owczarz and Z. Zlatev. Running a large-scale air pollution model on fast supercomputers. *Atmospheric Modeling*, to appear.

[13] A.S. Tomlin, S.Ghorai, G.Hart, and M.Berzins. 3-D adaptive unstructured meshes in air pollution modeling. *Environmental Management and Health*, volume 10, issue 4, 1999.