A Grid Solver for Reaction-Convection-Diffusion Operators[1]

Hatem Ltaief, Rainer Keller[2], Marc Garbey and Michael Resch[3]

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
`http://www.cs.uh.edu`

Technical Report Number UH-CS-07-08

July 19, 2007

Keywords: Grid solver, Domain Decomposition, Distributed computing, Load balancing

Abstract

In this paper, we present and analyze the performance of a fast parallel distributed computing time integration procedure for systems of Reaction-Convection-Diffusion equations. Typical applications include large scale computing of air quality models or population models in biology for which the main solver corresponds to a Reaction-Convection-Diffusion operator. One starts from a stabilized explicit time stepping scheme analyzed by Dupros et al (Int. Journal for Numerical Methods in Fluids, 2006). The numerical efficiency and the parallel scalability of this algorithm have been demonstrated on homogeneous parallel architectures. We introduce here an additional domain decomposition component to the algorithm to extend the scalability of the method to multi-cluster architectures. The targeted computer architecture is a high latency/low bandwidth network of few parallel systems. This paper provides one of the rare examples of a Partial Differential Equation application that is both numerically efficient and scalable on a wide area network of O(10) parallel systems.

[1]Research reported here was partially supported by Award 0305405 from the National Science Foundation.
[2]keller@hlrs.de, HLRS, University of Stuttgart, Germany
[3]resch@hlrs.de, HLRS, University of Stuttgart, Germany

A Grid Solver for Reaction-Convection-Diffusion Operators

Hatem Ltaief, Rainer Keller, Marc Garbey and Michael Resch

Abstract

In this paper, we present and analyze the performance of a fast parallel distributed computing time integration procedure for systems of Reaction-Convection-Diffusion equations. Typical applications include large scale computing of air quality models or population models in biology for which the main solver corresponds to a Reaction-Convection-Diffusion operator. One starts from a stabilized explicit time stepping scheme analyzed by Dupros et al (Int. Journal for Numerical Methods in Fluids, 2006). The numerical efficiency and the parallel scalability of this algorithm have been demonstrated on homogeneous parallel architectures. We introduce here an additional domain decomposition component to the algorithm to extend the scalability of the method to multi-cluster architectures. The targeted computer architecture is a high latency/low bandwidth network of few parallel systems. This paper provides one of the rare examples of a Partial Differential Equation application that is both numerically efficient and scalable on a wide area network of O(10) parallel systems.

Index Terms

Grid solver, Domain Decomposition, Distributed computing, Load balancing

I. Introduction

This paper focuses on the distributed computing of applications driven by the time integration of Reaction-Convection-Diffusion (RCD) system:

$$\frac{\partial C}{\partial t} = \Delta C + (\vec{a}(x,y,z,t).\nabla)C + F(x,y,z,t,C). \tag{1}$$

We have $C \equiv C(x,y,z,t) \in R^m,\ (x,y,z) \in \Omega \subset \mathbf{R}^3, t > 0$. A typical example is an air quality model where $\vec{a}$ is the given wind field, and F is the reaction term combined with source/sink terms. For such systems, m might be large and the corresponding Ordinary Differential Equation (ODE) system

$$\frac{\partial C}{\partial t} = F(x,y,z,t,C) \tag{2}$$

is stiff [1], [2]. A second class of examples is the time evolution of a population model either in ecology or biology [3], [4].

The equation in (1) can be rewritten as

$$\frac{DC}{Dt} = \Delta C + F(x,y,z,t,C), \tag{3}$$

where $\frac{D}{Dt}$ represents the total derivative.

There are a number of time integration schemes that have been applied to similar problems. We refer to [5], [2] for a review of these methods. A new time stepping scheme was introduced in [6] to simplify the parallel implementation of such Partial Differential Equation (PDE) problems and leads to a numerically efficient scalable solver. The main idea consists in stabilizing, with *a posteriori* filtering, the explicit treatment of the diffusion term. The diffusion term is a source term in the fast ODE solver and the problem is completely parameterized by space dependency. The coding of the algorithm is particularly simple and requires only a post-processing subroutine that relies on a fast trigonometric transform. The mathematical idea of the method was first introduced in [7] and was

Research reported here was partially supported by Award 0305405 from the National Science Foundation.
keller@hlrs.de, HLRS, University of Stuttgart, Germany
resch@hlrs.de, HLRS, University of Stuttgart, Germany

limited to regular grids. However, one can extend the method to situations where the discretization can be mapped to a regular space discretization or can be decomposed into subdomains with regular space discretization.

This paper introduces a Domain Decomposition (DD) that is very tolerant to high latency and low bandwidth networks. Each domain is solved by the original stabilized algorithm of Dupros et al. We present a performance analysis of the code, using as a computing platform a wide area network of few parallel systems.

We demonstrate that our algorithm is scalable for such a challenging computer environment. This paper provides, then, an example of a numerically efficient algorithm for a system of PDE that works well in grid computing and generalizes the result of [8] that applies only to elliptic operators.

The plan of this article is as follows. Section 2 presents our methodology for RCD problems and details the numerical algorithm. Section 3 presents the hardware/software infrastructure used in our distributed computing experiments. Section 4 discusses the partitioning of the data and the load balancing technique. Section 5 describes our experiments and presents our performance analysis. In Section 6, we give a synthesis of our results and comment on the possible generalization of our method.

II. Numerical Method and Algorithm

Let us introduce the semi-discretization in time of the PDE problem (1). For simplicity, the domain of computation Ω is the box $(0, L_x) \times (0, L_y) \times (0, L_z)$, discretized in space by a Cartesian grid. To take advantage of the fast solver described here, complex geometry might be addressed via immersed boundary technics [9], [10], [11]. We will consider the first order semi-implicit Euler scheme,

$$\frac{C^{n+1} - C^{n,*}}{dt} = \Delta C^n + F(x, y, z, t, C^{n+1}), \tag{4}$$

as well as the following second order scheme:

$$\frac{3C^{n+1} - 4C^{n,*} + C^{n-1,*}}{2\ dt} = 2\Delta C^n - \Delta C^{n-1} + F(x, y, z, t, C^{n+1}). \tag{5}$$

The equation in (5) is a combination of backward second order Euler (BDF) for the time derivative and second order extrapolation in time for the diffusion term. We recall that BDF is a standard scheme used for stiff ODEs [12], [13], [2].

$C^{n,*}$ is constructed with the method of characteristics [14] in order to include the convective term $(\vec{a}.\nabla)C$. To be more specific, let us consider the transport equation

$$\frac{\partial C}{\partial t} = a_1(x, y, z, t)\frac{\partial C}{\partial x} + a_2(x, y, z, t)\frac{\partial C}{\partial y} + a_3(x, y, z, t)\frac{\partial C}{\partial z}, \tag{6}$$

where $a_j,\ j = 1..3$ are the components of vector $\vec{a}$, and are function of (x, y, z, t). A first order approximation in time writes

$$C(x, y, z, t^{n+1}) = C(x - a_1^n\ dt, y - a_2^n\ dt, z - a_3^n\ dt, t^n).$$

It gives us in (4)

$$C^{n,*} = C(x - a_1^n\ dt, y - a_2^n\ dt, z - a_3^n\ dt, t^n).$$

Similarly, one can reconstruct a second order approximation in time of the C^* terms in (5).

The explicit diffusion term ΔC^n is discretized by a standard centered seven point formula.

The non linear equations of unknown C^{n+1} in (4) or (5) can be solved with a point-wise Newton procedure. These equations are parameterized by the space dependency thanks to the combination of the explicit treatment of the diffusion term and the use of the characteristic method for the convective term. The drawback of this time integration scheme is that we have a priori a severe time step restriction due to the stability condition generated by the explicit diffusion term. It was shown in [6] that one can overcome this difficulty by applying at each time step a stabilization procedure. We refer to that paper for the detailed description of the method as well as the verification of its numerical efficiency. The validation of the method was done with the benchmark problem of [15] as well as with a simplified ozone model that has four equations. Conceptually the stabilized scheme writes

$$C^n \xrightarrow{\mathcal{Q}} C_p^{n+1} \xrightarrow{\mathcal{S}} \tilde{C}_p^{*,n+1} \xrightarrow{\mathcal{P}_{sin}} \hat{C}_p^{n+1} \xrightarrow{\mathcal{D}_\sigma} \hat{C}_\sigma^{n+1} \xrightarrow{\mathcal{P}_{sin}^{-1}} \tilde{C}^{n+1} \xrightarrow{\mathcal{S}^{-1}} C^{n+1}.$$

$\mathcal{Q}$ is one of the time stepping schemes (4) or (5) and provides the prediction C_p^{n+1}. S is a shift that set up homogeneous boundary conditions thanks to an affine point-wise transformation. The filter D_σ cuts out all frequencies that do not satisfy the Von Neumann criterion of stability. To compute those frequencies, we use $\mathcal{P}_{sin}$ that computes a three dimensional trigonometric expansion of the function $C(x, y, z, t)$ at some given time t. The transformation $\mathcal{P}_{sin}^{-1}\ o\ D_\sigma\ o\ \mathcal{P}_{sin}$ can be processed as a sequence of Kronecker products.

The structure of the parallel algorithm is then as follows.

The code calculates a four dimensional array $C(1:m, 1:Nx, 1:Ny, 1:Nz)$ where the first index corresponds to the dimension m of the chemical species, the second, third and fourth indices correspond to space dependency. For the sake of simplicity, the data are distributed on a two dimensional grid of processors of dimension $px \times py$ that matches the space domain in (x,y) directions. Each processor owns a column in the z direction of all C components. The computation is decomposed into two phases:

- Step 1: Solution of the formula with a Newton loop

$$C(:,i,j,k) := Q(C(:,i,j,k), C(:,i+1,j,k), C(:,i-1,j,k), C(:,i,j+1,k),$$
$$C(:,i,j-1,k), C(:,i,j,k+1), C(:,i,j,k-1)),$$

  at each grid point with appropriate boundary conditions.
- Step 2: Evaluation of the filtered solution $C(:,i,j,k)$ using the stabilization technique.

The parallel implementation of Step 1 is straightforward. If the load per processor is high enough, which is likely to be the case with the pointwise integration of the nonlinear set of equations, this algorithm scales very well (i. e., see for example the parallel CFD test case in `http://www.parcfd.org`). To simplify the presentation, we assume that the number of Newton iterations is uniformly the same. But this may not be true in general and can impact the load balancing with the data distribution.

The data structure is imposed by Step 1. Step 2 introduces a global data dependency across i, j and k. It is therefore more difficult to parallelize the filtering algorithm. The kernel of this algorithm is to construct the three dimensional sine expansion of $C(:,i,j,k)$ modulo a shift, and its inverse. One may use an off the shelf parallel FFT library that supports multi-dimension distributions of matrices (e. g., `http://www.fftw.org`). In principle, the arithmetic complexity of this algorithm is of order $m\,N^3\log(N)$ if $Nx \sim N$, $Ny \sim N$, $Ny \sim N$. It is well known that the inefficiency of the parallel implementation of the FFTs comes from the global transpose of $C(:,i,j,k)$ across the two dimensional network of processors. In this case, an alternative approach to FFTs that takes full advantage of the vector data structure of $C(:,i,j,k)$ is to write Step 2 in matrix multiply form. Since $C(:,i,j,k)$ is distributed on a two dimensional network of processors, one can use an approach very similar to the systolic algorithm to perform in parallel the matrix multiply. It was shown in [6] that the matrix multiply algorithm has good scalability properties on a homogeneous parallel computers equipped with a fast network. However, we will see that this algorithm is fairly inefficient if the network of the parallel system is slow. The data distribution of this first part of our algorithm can be seen as a non-overlapping DD that uses ghost cells for message passing between processors.

We introduce then a second level of parallelism by first decomposing the domain Ω into a set of nd overlapping subdomains $\Omega_k = (X_k^l, X_k^r) \times (0, L_y) \times (0, L_z),\ k = 1..nd$. We have

$$X_1^l = 0 < X_2^l < X_1^r < X_3^l < X_2^r < \ldots < X_{nd} = L_x. \qquad (7)$$

Each domain Ω_k is processed on a homogeneous parallel system with the stabilized algorithm described above. At the end of each time step, these parallel systems exchange the boundary conditions at the interface exactly as in the Additive Schwarz algorithm [16]. It was shown experimentally in [7], provided that the overlap is large enough to damp all unstable frequencies, one gets a stable time stepping scheme. We refer to [17] for the general analysis of explicit DD methods based on implicit subdomain solver for parabolic operators. Overall, we combine then a first level of parallelism corresponding to the stabilized time marching procedure with a second level of parallelism that uses overlapping DD. We will refer to the subdomains of this second level of DD as macro subdomains. Thanks to this second level of DD, we can keep the dimension of macro subdomains of order 100 or less in each space direction and have the matrix multiply approach used inside macro subdomains fairly competitive versus the FFT. The data distribution and the topology of communication of processors will be covered in more details in Section IV.

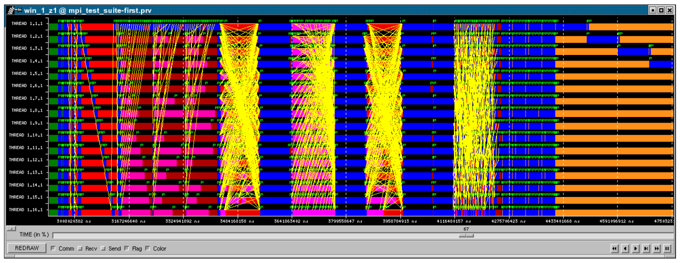

Fig. 1. Paraver trace of the communication patterns of an MPI testsuite.

We are now going to describe the computer hardware/software infrastructure set up for the performance analysis of the DD algorithm.

III. Hardware/Software Infrastructure

Our experiment relies heavily on a message passing library designed specifically for distributed computing on wide area network. We will next describe briefly the basic principle of this middleware.

A. *PACX-MPI Installation*

The computational grid library PACX-MPI [18] allows MPI-parallel applications to run on a distributed set of computational hardwares in a seamless way by abstracting the calls to MPI-functionality. One need only recompile the application code. Further it is not necessary to make the application aware of the underlying topology. However, it is indeed preferrable to take advantage of that knowledge for performance tuning reasons. Communication is done through two MPI processes, called daemons, which handle the communication over an interconnecting network using some underlying protocol, e.g., TCP, IMPI based on TCP, SSL based on TCP or UDP. Using either protocol, the interconnecting network is the bottleneck for computation. One therefore would like to minimize the performance impact of communication between interconnected machines. In this work, we will describe the novel work on performance analysis with PACX-MPI distributed applications using the Paraver [19] performance tool.

In the following, we will use the term ”local” functions to denote routines which only involve the calling process. Similarly, routines are ”non-local” functions when they require the calling on other processes as well (e.g., the collective routines in MPI).

PACX-MPI works by replacing every non-local calls to MPI as well as some local calls to MPI by a call to the equivalent PACX-routine. For initialization, instead of `MPI_Init` the function `PACX_Init` is called. The main MPI-structures such as communicators, groups and datatypes have to be wrapped as well to map information regarding external datatype-representation and transcode for sending/receiving to other machines in case of heterogenous setups.

Paraver trace-files are gathered by linking the application with the `mpitrace`-library, which uses the profiling interface `PMPI` to wrap MPI-functions. At entry and exit of, e.g., `MPI_Recv` the entry time and message envelope, i.e., the tuple $(destination, tag, communicator)$ plus information regarding the size of the communication is stored in the calling process' trace file; at the exit of this function the time again is stored. After the execution, all processes' trace files are merged into one single Paraver file. Figure 1 shows the Paraver window with the main window showing the trace of an MPI testsuite. Different point-to-point communication patterns between the eight processes are visible: e.g., nearest-neighbor, many-to-one and all-to-all communication may be distinguished.

To perform analysis of the communication pattern of PACX-MPI enabled applications, the `mpitrace` was extended within the HPC-Europa project [20]. As `mpitrace` uses the PMPI-interface (as does PACX-MPI for Fortran applications), special action had to be used to get the linking order of the application right. Instead of generating one specifc Fortran naming interface in the static library for the Fortran version of `libpacxf`, we used the weak symbols of the linker to have functions both in PACX-MPI and the tracing library, e.g., `mpi_irecv_` in `libpacxf` and `pacxtracef`, respectively. Figure 2 shows the linking order of Fortran applications linked against the `libpacxtracef`, `libpacxf`, `libpacx` and finally `libmpi`.

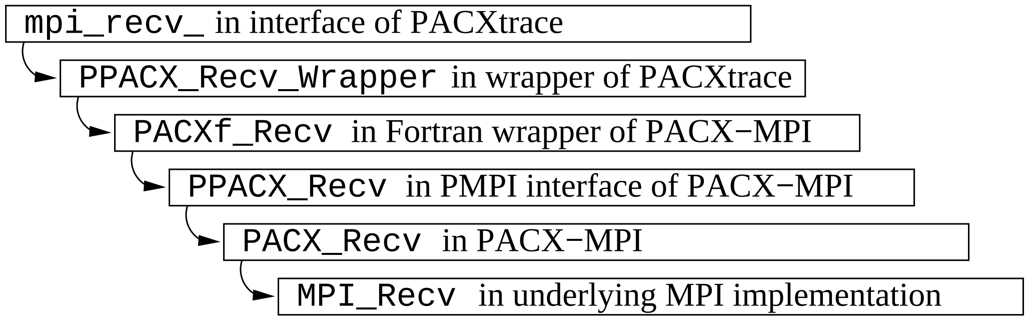
mpi_recv_ in interface of PACXtrace
PPACX_Recv_Wrapper in wrapper of PACXtrace
PACXf_Recv in Fortran wrapper of PACX−MPI
PPACX_Recv in PMPI interface of PACX−MPI
PACX_Recv in PACX−MPI
MPI_Recv in underlying MPI implementation

Fig. 2. Linking order / calling stack of PACX-MPI applications to be traced.

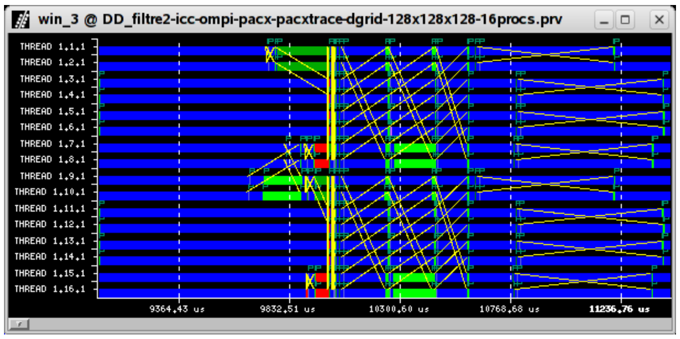

Fig. 3. Zoom into one iteration of the solver.

With a domain-size of 128^3 split onto two macro subdomains, the time for computing the inner domain is around 900 microseconds per processor. The heavy communications within the macro subdomains consist of large messages, while inter-macro subdomain communications consist of $O(n)$ sends/recvs small communications (63kB) with n being the number of macro subdomain boundary processors. Figure 3 shows the trace with 16 processes on the global domain.

Let us describe now the computer architectures used in our grid computing experiments.

B. Overview of the Grid Infrastructure

Our test grid is composed mainly of three clusters: *atlantis.tlc2.uh.edu* at the University of Houston (TX, USA), *cacau.hww.de* at the University of Stuttgart (HLRS, Germany) and *cluster150.inm.ras.ru* at the Institute of Numerical Mathematics in Moscow (Russian Academy of Sciences, Russia). The rationale behind the choice of this configuration is as follows. First, we had access to all three systems for an extended period of time on the order of a few months and we were able to install PACX-MPI on our user accounts. Second, we were able to reserve a number of processors on each system with no time sharing for the duration of each experiment, that is, on the order of a few hours. Third, while this configuration was limited to three systems, this specific grid is completely heterogeneous and has a fairly slow interconnect network. We claim, then, that the configuration of computers described below is representative of a standard industry network that is interested in using its existing distributed resources to run larger simulations rather than on any individual system.

Table I shows a general hardware description of each machine. Furthermore, the machines have different processor, memory, and network characteristics, which actually correspond to a typical grid environment. Concerning the message passing software, we use on each machine the MPIch-1.2.5 [21] and the PACX-MPI-5.0-library, compiled with the Intel compiler version 9.0.
To document our performance analysis of our DD code, we first measured the performance of the internal network of each parallel system. We use the *iperf* [22] and *traceroute* [23] tools, respectively to compute the bandwidth and latency of the network. *Cacau* achieves a bandwidth of $930 - 950 Mbits/sec$ and a latency of $0.7ms$, *Atlantis*

Name	Nodes	CPU Type	Memory	Network Interconnect
Atlantis	152	Dual Itanium2 $1.3GHz$	4 GB	Gigabit and Myrinet
Cacau	204	Dual Xeon EM64T $3.2GHz$	1 or 2 GB	Gigabit and Infiniband
Cluster150	16	Dual Itanium2 $1.6GHz$	4 GB	Gigabit and Myrinet

TABLE I

MACHINE DESCRIPTIONS.

Machine Name	Grid Size	10000	15000	25000
Atlantis	1×8	0.7165	0.7418	0.7723
	2×4	0.875	0.819	0.812
	4×2	1.03	0.865	0.698
	8×1	0.9371	0.958	0.974
Cacau	1×8	2.304	2.403	2.486
	2×4	2.365	2.44	2.506
	4×2	2.333	2.397	2.482
	8×1	2.095	2.228	2.337
Cluster150	1×8	0.8795	0.8942	0.9033
	2×4	1.066	0.979	0.952
	4×2	1.27	1.135	0.864
	8×1	1.173	1.151	0.8911

TABLE II

HPL BENCHMARK RESULTS (GFLOPS/S) ON ATLANTIS, CACAU AND CLUSTER150 WITH GIGABIT ETHERNET INTERCONNECTS.

a bandwidth of $910 - 930 Mbits/sec$ and a latency of $0.09ms$ and finally, *Cluster150* develops a bandwidth of $920 - 940 Mbits/sec$ and a latency of $0.08ms$. The main difference between each system relies then on the latency performance.

Second, we assessed the flops performance of each system. Table II recalls the High Performance Linpack (HPL) benchmark for distributed memory computers. The HPL benchmark is a portable software package that solves a dense linear system with the LU decomposition algorithm. More details on this benchmark can be found in [24]. The different grid sizes have been chosen to insure that the entire data of the benchmark, mainly the dense matrix, fit inside the main memory. We thus avoid any expensive swapping with the disk space.

We can see a fairly large heterogeneity in performance of these three systems. *Cacau* seems to be the fastest machine reaching up to $2.5GFlops/s$, while *Cluster150* and *Atlantis* are roughly two to three times slower.

This disparity in performance can be explained in part by a comparison of all three architectures on the efficiency of the memory access. We have measured the access to memory efficiency with STREAM [25], a small program which measures the access memory performance of four long vector operations. Table III evaluates the bandwidths to perform the different arithmetic operations on each system. Again, the memory access speed is not the same from one machine to another.

In addition to the internal network and flops performance disparities of each system, we will show that the wide area network performance between each system adds another level of heterogeneity in the grid environment.

Operation	Copy	Scale	Add	Triad
Atlantis	3.4	3.4	3.8	3.8
Cacau	2.1	2.1	2.5	2.5
Cluster150	3.6	3.6	4.1	4.1

TABLE III

STREAM BENCHMARK RESULTS IN GBITS/S.

Bonds	Bandwidth (Kbits/sec)	Latency (ms)
Atlantis-Cacau	$304 - 373$	$62.5 - 66.4$
Cacau-Cluster150	$546 - 808$	$27.3 - 29.3$
Cluster150-Atlantis	$71 - 127$	$83 - 84$

TABLE IV
THE NETWORK PERFORMANCE BETWEEN MACHINES.

C. *Network Evaluations*

Figure 4 shows the grid set up with the three computers described previously. One use in particular, the two additional daemons from PACX-MPI to perform message passing between and within each machine.

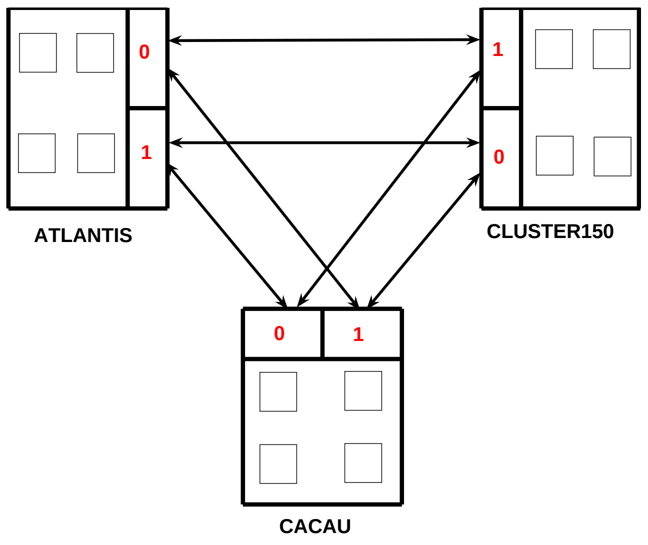

Fig. 4. The Grid Infrastructure.

The network performance at the time of our simulation was determined with the *Ping-Pong* benchmark, which evaluates point-to-point message passing operations (a *send* and a *recv*) on a pair of processes. Table IV gives the minimal and maximal Ping-Pong bandwidth and latency.

This low performance of the network between sites is in general the main limitation to grid computing efficiency. Many applications require heavy communications between peers and therefore cannot run properly on such a grid environment. It is certainly true that higher bandwidth can be achieved with special connections between remote sites but it is unrealistic to assume better latency time from one system to another when these systems are hosted by individual laboratories, as it is often the case. We have then to deal with networks in grid computing that have latency of the order of 1000 times larger than standard parallel systems. This motivates the original algorithm development of this paper.

We will now discuss the load balancing aspect of our simulation that is required by the heterogeneity of this experimental grid of computers.

IV. DATA DISTRIBUTION AND LOAD BALANCING

As seen in the hardware descriptions of section III-B, the machines are quite heterogeneous and one should pay attention to how the application load must be distributed across the peers to get the best optimized execution time. We present here a simple and rapid load balancing procedure that takes advantage of the strip DD of the three dimensional regular Cartesian grid that we keep. The DD is performed only in the x space direction. The dimension of the grid in y and z direction is fixed once and for all. We recall that those domains are macro subdomains that are themselves processed in parallel. Let us assume that we have nd macro subdomains distributed among q parallel systems. As we will see in our numerical experiments, the two step stabilized time integration scheme has very poor performance if it is run across the wide area network. We assume then that no macro subdomains are shared by two parallel systems. Each macro subdomain processed by a parallel system is decomposed further into a two dimensional grid of non-overlapping subdomains processed in parallel by the stabilized time stepping procedure. We have then a two level DD that is designed to match the hierarchy of memory access of a cluster of clusters. Figure 5 gives an example of this two level DD with our three parallel computers: from left to right, Atlantis, Cacau and Cluster150. The black thick and dashed lines delimitate the different subdomains or pencils. The red lines demarcate the two macro subdomains within a single machine while the diamond lines represent the macro subdomains virtual boundaries handled between two neighbor remote sites.

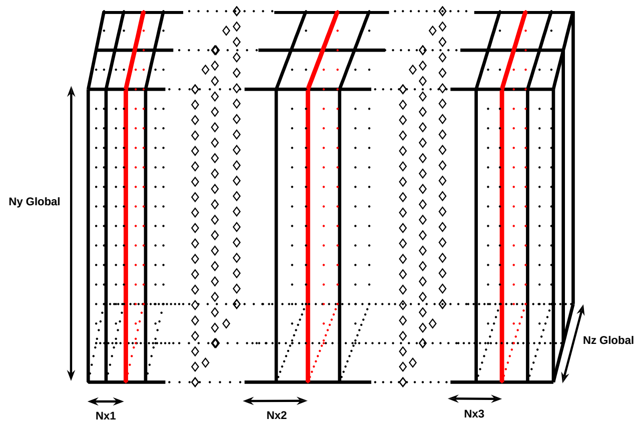

Fig. 5. DD with six macro subdomains, two on each machine.

The overlap between macrodomains is fixed once and for all experiments according to the stability of the time step procedure [7].

Let us denote

$$l_k = \frac{X^l_{k+1} + X^r_k}{2}, \; k = 1..nd,$$

a partition of the interval $(0, L_x)$ associated with the overlapping macro subdomain DD (7). We assume that the elapsed time to compute each macro subdomain for a given data set of artificial boundary conditions is a function of the number of grid points only. This is a realistic hypothesis when the number of Newton loops in Step 1 of the stabilized algorithm is fixed. It is in practice true for a period of time with a duration that depends on the dynamic of the reaction term, and can be verified a posteriori. Let us denote $G_n : \alpha \rightarrow t_{elapsed}$ the function that gives the elapsed time to process on the computer $n \in (1..q)$, one time step of the RCD system in one macro subdomain of dimension $\alpha\ L_x, \alpha \in (0, 1)$, in x direction. In our DD, we have $\alpha_k = l_{k+1} - l_k + d$, where d is the size of the overlap in x direction. $G_n(\alpha_k)$ might be a monotonically increasing function of α_k if there is no significant cache memory effect.

Because communications between macro subdomains are limited to one exchange of interfaces at the end of each time step between neighbor processors, load balancing should be achieved, in first approximation, if $G_n(\alpha_k),\ k = 1..nd$ is the same across all parallel systems, $n = 1..q$.

The load balancing problem is equivalent to

$$\text{Minimize} \sum_{k=1...nd-1} (G(\alpha_{k+1} - G(\alpha_k))^2 \tag{8}$$

under the constraints:

$$\sum_{k=1}^{nd} (\alpha_k - d) = 1 \text{ and } \alpha_k \geq d,\ \forall k = 1..nd \tag{9}$$

To ease the solution of the optimization problem (8,9), one computes a least square approximation of the elapsed time $G_n(\alpha)$ with a quadratic polynomial:

Name	Grid Size	Nx_{local}
Atlantis	T_1	76
	T_2	112
Cacau	T_1	122
	T_2	182
Cluster150	T_1	90
	T_2	132

TABLE V
LOCAL SUBDOMAIN GRID SIZES IN x DIRECTION.

$$\tilde{G}_n(\alpha) = a_n \alpha^2 + b_n \alpha.$$

This approximation is justified by the fact that the arithmetic complexity of the matrix multiply procedure in Step 2 is quadratic, while the arithmetic complexity of everything else is linear. Further, the elapsed time should be null for $\alpha = 0$.

This estimate presents several advantages:

(1) $G_n(\alpha)$ can be computed with embarrassing parallelism prior to the simulation, and with no communication between each remote site.

(2) It can be verified with a standard statistical test of the least square fit accuracy.

(3) The overall optimization solution of problem (8,9) where G has been substituted by $\tilde{G}$ might be found extremely quickly with a method of descent.

One can then dynamically reset the load balancing every time the number of Newton loops in each macro subdomain starts to evolve significantly. The new DD should involve message passing between ”neighbor” parallel systems of data set with dimensions $|\alpha_k^{new} - \alpha_k^{old}|\ L_x\ L_y\ L_z$. Otherwise static load balancing, i.e., once and for all setting up the α_n partitioning, would be enough.

We have validated this procedure with our RCD code for the simplified ozone model of [6] and the three parallel systems configuration described previously. We consider two test cases denoted T_1 and T_2 with global grid size $(Nx, Ny, Nz) = (282, 96, 96)$ and $(Nx, Ny, Nz) = (420, 96, 96)$, respectively. Table V shows the different loads per subdomain with two overlapping meshes included.

We can then compute a posteriori in the simulation the idle time and verify a posteriori the quality of the load balancing. Next, we will present the results of our experiments.

V. EXPERIMENTAL RESULTS

Our experiments have been designed to answer the following two questions:

- Q1: Does our simulation scale with the number of processors on our network of parallel systems?
- Q2: How much performance do we yield by using the grid instead of a single parallel system?

The rationale behind Q1 is that grid computing allows, in principle, the solving of a larger problem than one single computer of the grid. But we need to verify that the elapsed time does not increase dramatically with the number of subdomains. Q2 refines the analysis by assessing what is the real overhead of grid computing versus traditional computing. We will try to answer these questions for an algorithm that has been specifically designed for the grid computing environment.

We recall that we have obtained excellent speedup and scalability for our method on a homogeneous parallel system in [26]. For completeness, let us summarize this result. The tests were all done on a Cray T3E/512, an MPP installed at the High Performance Computing Center at Stuttgart, HLRS. The architecture of this machine is well-known and offers a balanced communication and computation performance. Figure 6 shows the efficiency of the parallelization according to the domain decomposition with growing size, i. e., $124 \times 258 \times 32$ for curve ’o’, $244 \times 258 \times 32$ for curve ’+’, $484 \times 258 \times 32$ for curve ’*’ and $960 \times 128 \times 32$ for curve ’v’. Figure 7 shows the elapsed time in seconds for 5 configurations with increasing number of subdomains nd: curve ’o’ respectively ’+’, ’*’, ’d’ ’v’ for growing sizes from $nd = 1$, to $nd = 5$. We can observe that (1) the larger the number of subdomains, the better the scalability, and (2) the domain decomposition has a super-linear speed-up.

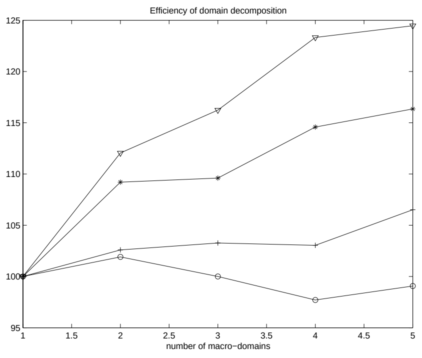

Fig. 6. Parallel efficiency with respect to the number of subdomains.

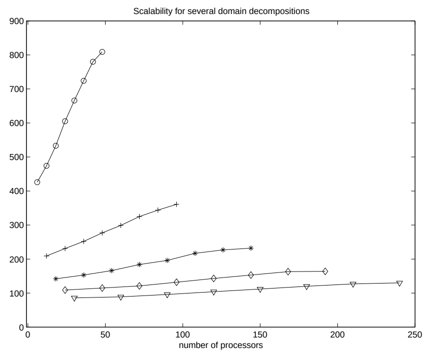

Fig. 7. Scalability for several domain decomposition configurations.

Number of Subdomains	3	6	12	24
Small	15.4	14.2	10.2	3.3
Large	11.6	12.2	9	4.4

TABLE VI
OVERHEAD IN PERCENTAGE.

For our grid experiments, we will keep two subdomains per each macro subdomain solved by two processors, and have the size of the macrodomain constant with the test cases T1 and T2 load balanced as in Section IV. We will test the scalability and performance of our code with 3, 6, 12, and finally 24 macro subdomains respectively.

To address Q1, figures 8 and 9 show a nice scalability of our simulation with the grid set up described previously. These figures give the elapsed time for runs with 8 time steps and $d = 2$ overlapping meshes. For the T1 test case, we have 350 K grid points per processor for Atlantis, 562K for Cacau, and 415K for Cluster150. These numbers of grid points are respectively 516K, 839K and 608K for the test case T2. We test then the efficiency of our method with a relatively small problem. This is more challenging indeed, but fits better the application requirement.

For each vertical bar, we have stacked the long distance communication above the computation times. So, we can see that the communication time between remote sites is fairly low compared to the time spent in computing. While the computation is fairly intensive inside the macro subdomains, the amount of communications between two neighbors stays low. The dimension of the interfaces to be sent is $d \times N_y \times N_z$ and is identical for both test cases T1 and T2. *Atlantis* and *Cluster150* are sending and receiving 2.46 Mb, while Cacau, at the *center* of the grid topology of communication, is sending and receiving two times this size, i.e., 4.92 Mb. Table VI demonstrates that the overhead induced by the distanced inter-communications is small. Further, we observe that the more subdomains, the lower the inter-communication overhead.

Let us underline that this overhead will vary indeed with the ratio of the dimension of the interface $d \times N_y \times N_y$ versus the size of the macro subdomain $\alpha N_x \times N_y \times N_z$.

To emphasize these first results, we decided to experiment the effect of a wrong DD on the general peformance by splitting a single macro subdomain between two parallel computers, Atlantis and Cacau. Figures 10 and 11 give the execution time in seconds, according to a correct DD or a wrong DD, with T1 and T2 test cases. The overhead induced is very significant: a growing factor of 4 and 3 for T1 and T2, respectively. These intensive inter-communications, originally performed within a machine, appear now to be a bottleneck compared to the conclusions from figures 8 and 9.

To address question Q2, we have shown in figures 12 and 13 the execution time with the same DD as in the grid experiments, but with *all* processes running on a single parallel computer. The dimension of the subdomains is chosen as in T1 or T2, depending on the system used. The overall grid size of the problem then is not constant. Consequently we have chosen to run the test cases with the computers of the grid that have the best and the worst performance for our code, respectively *Cacau* and *Atlantis*.

As expected, internal inter-communications can be completely neglected with regard to the computation time for both computers. Moreover, we observe that the execution time when running only on *Cacau* corresponds to 65% of

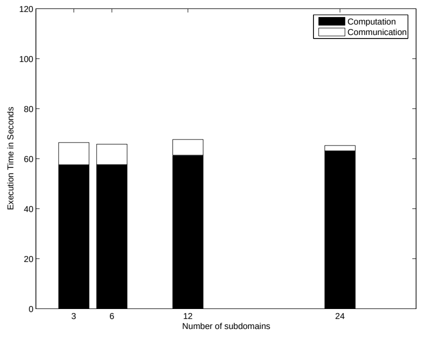

Fig. 8. Grid Experiment with T1.

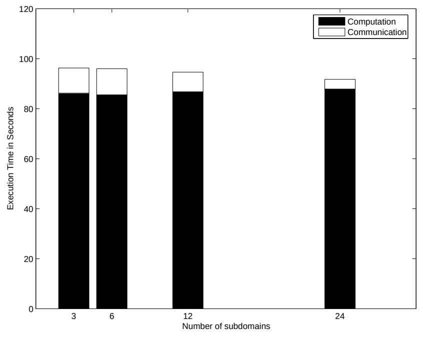

Fig. 9. Grid Experiment with T2.

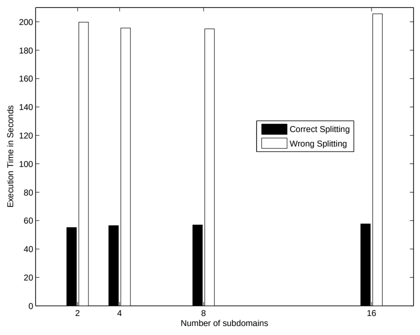

Fig. 10. Correct Versus Wrong Splitting with T1.

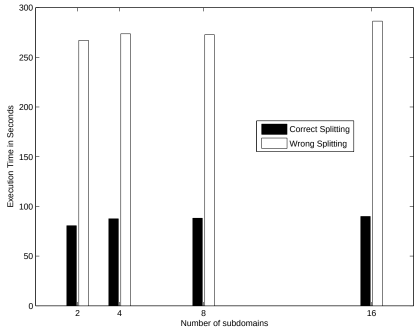

Fig. 11. Correct Versus Wrong Splitting with T2.

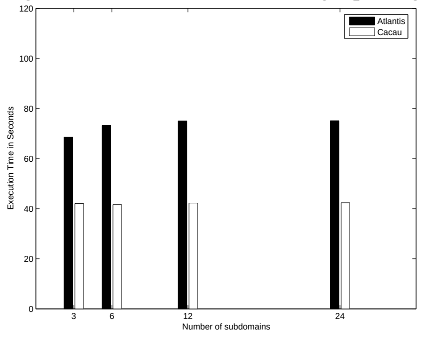

Fig. 12. All in one: Scalability with T1.

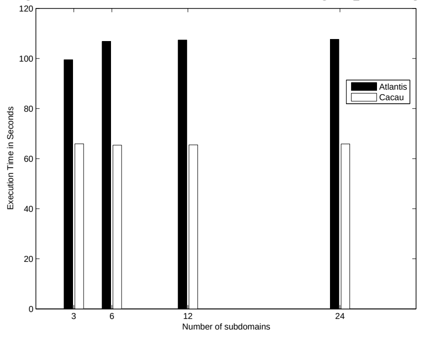

Fig. 13. All in one: Scalability with T2.

the overall execution time obtained in the grid set up with either small or large grid size. However, when running on the slowest machine, *Atlantis*, the execution time obtained on the grid for the same configurations are roughly 3 to 15% less than the *Atlantis*'s execution time.

The main idea here is not to say that our application performs better in grid computing environment than on a single parallel machine but rather to highlight the fact that, although grids are subject to heavy constraints such as very slow interconnect network, the RCD application is still able to run efficiently thanks to the salient feature of our method. We will now conclude our study.

VI. Conclusion

We have presented and analyzed in this paper, the performance of a fast parallel distributed computing time integration procedure for systems of RCD equations. Applications that can benefit from this algorithm are, for example, large scale computing of air quality models or population models in biology. We have used a stabilized explicit time stepping scheme analyzed by Dupros et al, as published in the Int. Journal for Numerical Methods in Fluids in 2006 [6], and have appended an additional DD component to the algorithm to extend the scalability of the method to multi-clusters architecture and/or grid computing. The targeted computer architecture is a high latency/low bandwidth network of few parallel systems. While our paper was limited to experiments with three parallel systems located in Russia, Germany and the United States, we believe that our method provides one of the rare examples of PDE applications that is both numerically efficient and scalable on a wide area network of O(10) parallel systems. To make it more practical, for long time numerical simulations that are actually typical of air quality or climate modeling, we need to make our simulation *fault tolerant*. It is realistic to assume that the network between remote computing sites will fail regularly. We refer to [27] for a preliminary algorithmic work along these lines.

References

[1] D. Dabdub and J. H. Steinfeld, "Parallel computation in atmospheric chemical modeling," *Parallel Computing*, vol. 22, pp. 111–130, 1996.

[2] J. G. Verwer, W. H. Hundsdorfer, and J. G. Blom, "Numerical time integration for air pollution models," Tech. Rep., Nov. 27 1998. [Online]. Available: http://citeseer.ist.psu.edu/365307.html;http://www.cwi.nl/ftp/CWIreports/MAS/MAS-R9825.ps.Z

[3] S. Dougall, A.R.A.Anderson, M.A.J.Chaplain, and J.A.Sherratt, "Mathematical modelling of flow through vascular networks: Implication for tumor induced angiogenesis and chemotherapy strategies," *Bulletin of Mathematical Biology*, vol. 64, pp. 673–702, 2002.

[4] K.J.Painter, N.J.Savill, and E.Shochat, "Computing evolution in brain tumours." [Online]. Available: http://www.ma.hw.ac.uk/~painter/publications

[5] D. Knoll and D. Keyes, "Jacobian-free Newton–Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, pp. 357–397, 2004.

[6] F. Dupros, M. Garbey, and W. E. Fitzgibbon, "A filtering technique for system of reaction diffusion equations," *Int. J. for Numerical Methods in Fluids*, vol. 52, pp. 1–29, 2006.

[7] M. Garbey, H. G. Kaper, and N. Romanyukha, "A some fast solver for system of reaction-diffusion equations," *Domain Decomposition Methods in Science and Engineering CINME*, pp. 387–394, 2002.

[8] N. Barberou, M. Garbey, M. Hess, M. M. Resch, T. Rossi, J. Toivanen, and D. Tromeur-Dervout, "Efficient metacomputing of elliptic linear and non-linear problems," *J. Parallel Distrib. Comput*, vol. 63, no. 5, pp. 564–577, 2003. [Online]. Available: http://dx.doi.org/10.1016/S0743-7315(03)00003-0

[9] P. Angot, C.-H. Bruneau, and P. Fabrie, "A penalization method to take into account obstacles in incompressible viscous flows," *Numerische Mathematik*, vol. 81, no. 4, pp. 497–520, 1999. [Online]. Available: http://link.springer-ny.com/link/service/journals/00211/bibs/9081004/90810497.htm;http://link.springer-ny.com/link/service/journals/00211/papers/9081004/90810497.pdf

[10] E.Arquis and J. Caltagirone, "Sur les conditions hydrodynamiques au voisinage d'une interface milieu fluide-milieux poreux: Application a la convection naturelle," *CRAS Paris II*, vol. 299, pp. 1–4, 1984.

[11] M.Garbey and F.Pacull, "A versatile incompressible navier stokes solver for blood flow application," *Int. J. of Numerical Method in Fluids*.

[12] L. Petzold, "Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 1, pp. 137–148, Mar. 1983.

[13] A. Sandu, J. G. Verwer, F. A. Potra, D. Dabdub, and J. H. Seinfeld, "Benchmarking stiff ODE solvers for atmospheric chemistry problems I: Implicit versus explicit," Tech. Rep., Oct. 29 1997. [Online]. Available: http://citeseer.ist.psu.edu/254311.html;http://www.math.uiowa.edu/~potra/REPORTS/rep85_96.ps.gz

[14] O. Pironneau, *Finte Element Methods for Fluids*. New York, New York: John Wiley & Sons, 1989.

[15] D. L. Ropp, J. N. Shadid, and C. C. Ober, "Studies of the accuracy of time integration methods for reaction-diffusion equations," *J. Comput. Phys.*, vol. 194, no. 2, pp. 544–574, 2004.

[16] P. L. Lions, "On the Schwarz alternating method I," in *Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, Eds. Philadelphia: Society for Industrial and Applied Mathematics, 1988, pp. 1–42.

[17] Y. Zhuang and X.-H. Sun, "Stable, globally non-iterative, non-overlapping domain decomposition parallel solvers for parabolic problems," in *SC*, 2001, p. 19. [Online]. Available: http://doi.acm.org/10.1145/582034.582053

[18] R. Keller, E. Gabriel, B. Krammer, M. S. Müller, and M. M. Resch, "Towards efficient execution of MPI applications on the Grid: Porting and Optimization issues," *Journal of Grid Computing*, vol. 1, no. 2, pp. 133–149, 2003.

[19] "Paraver Homepage," WWW, May 2006, http://www.cepba.upc.es/paraver.

[20] "HPC-Europa Homepage," WWW, June 2007, http://www.hpc-europa.org.

[21] W. Gropp and E. Lusk, "Installation guide to mpich, a portable implementation of MPI," May 05 2001. [Online]. Available: http://www.osti.gov/servlets/purl/378911-4SEje1/webviewable/

[22] C.-H. Hsu and U. Kremer, "IPERF: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation*, 1998. [Online]. Available: http://www-cse.ucsd.edu/$\sim$calder/fdo/archive/fdo1/papers/pfdc-hsu.ps.Z

[23] S. Branigan, H. Burch, B. Cheswick, and F. Wojcik, "What can you do with traceroute?" *IEEE Internet Computing*, vol. 5, no. 5, p. 96, 2001. [Online]. Available: http://www.computer.org/internet/ic2001/w5096abs.htm

[24] http://www.netlib.org/benchmark/hpl/.

[25] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computing," http://www.cs.virginia.edu/stream/.

[26] M.Garbey, R.Keller, and M.Resch, "Toward a scalable algorithm for distributed computing of air-quality problems," vol. Volume 2840/2003, pp. 667–671, 2003.

[27] H. Ltaief, M. Garbey, and E. Gabriel, "Parallel fault tolerant algorithms for parabolic problems," in *Lecture Notes in Computer Science, Proceedings of the 12th International Euro-Par Conference*, vol. 4128. Springer Berlin / Heidelberg, 2006, pp. 700–709.