Using SBASCO to Solve Reaction-Diffusion Equations in Two-Dimensional Irregular Domains

Manuel Díaz, Sergio Romero, Bartolomé Rubio,
Enrique Soler, and José M. Troya

Department of Languages and Computer Science, University of Málaga, 29071 SPAIN
{mdr, sromero, tolo, esc, troya}@lcc.uma.es

Abstract. The SBASCO programming environment provides the developer of parallel and distributed applications with high-level programming capabilities. This is achieved as a result of the combination of two technologies: algorithmic skeletons and software components. This paper is a case study on the use of SBASCO. Specifically, we present a scientific application to study the propagation of reaction waves in two-dimensional irregular domains which can be divided into overlapping rectangular regions. Domain decomposition techniques are used to solve a system of two non-linear reaction-diffusion equations. The structure of the application is established by means of a high-level skeleton, which captures all the communication and synchronization details that take place in parallel component interaction, thereby releasing the programmer from coding them. In addition, the use of software components facilitates the development process and allows the creation of more flexible and adaptable software.

1 Introduction

Domain decomposition techniques have received great attention especially for the numerical solution of partial differential equations (PDEs) [8]. In our context, the term "domain decomposition" means the separation of the physical domain into different regions and so, many of these techniques accept a parallelization that allows them to tackle large-scale and realistic engineering problems [7].

This paper presents a practical case study using SBASCO in the development of a parallel scientific application in order to obtain the numerical solution of a reaction-diffusion problem. The problem is modelled as a system of two time-dependent, nonlinearly coupled PDEs, and is solved by means of domain decomposition methods. Two-dimensional domains that exhibit an irregular geometry with re-entrant corners are considered. Overlapping domain decomposition techniques used in this paper are based on Schwarz's methods which, at the differential level, use the solution in one subdomain to update the Dirichlet data of the other, the convergence rate being influenced by the overlapping length [5].

The application combines two different parallelism levels. On the one hand, the solution in the different subdomains is computed in parallel by assigning

one task per subdomain. On the other hand, the numerical method each task implements, which is basically a procedure for solving a large linear system of algebraic equations, may be parallel itself, e.g. data parallel red-black Gauss-Seidel relaxation is used in this work. Moreover, the updating of the interior boundaries of adjacent subdomains, which takes place in every iteration, involves the communication and synchronization among either sequential or parallel tasks. These communications depend on the data distribution belonging to the participant tasks. The development of such applications can be tedious and error-prone when based on traditional high-performance solutions such as HPF/MPI, C/PVM, etc. and so, we encourage the use of languages and tools that offer a higher degree of abstraction.

SBASCO (Skeleton-Based Scientific Components) [2] is a new programming environment focused on the efficient development of parallel and distributed numerical applications, also integrating two different technologies: algorithmic skeletons [1] and software components [4]. This unified approach provides interesting features in terms of interoperability, high-level programmability, compositionality, and code reusability.

The *multiblock* skeleton defined in SBASCO captures the pattern of parallel computation and communication that takes place in the proposed application. This skeleton allows the establishment of the application structure in an elegant and declarative way, and also abstracts the programmer from most of the low-level aspects of parallelism exploitation such as the creation, communication and synchronization of tasks. They are addressed by the runtime support which implements the skeleton so that the programmer can focus on writing the scientific code, which is encapsulated into software components.

This paper is structured as follows. The next section outlines the main characteristics of SBASCO. Section 3 introduces the physical problem taken into consideration. The design of the application, including some implementation details are described in section 4. Experimental results are shown in section 5. Finally, some conclusions are outlined.

2 SBASCO Overview

The skeleton-based composition language of SBASCO is used for both scientific components and application construction. The internal structure of a component can be established by means of the application of a fixed set of skeletons. So, the interaction of the different tasks integrating the components is expressed in a high-level and declarative way, according to static and predictable patterns. The following is a brief description of the skeletons provided:

- The *multiblock* skeleton is focussed on the solution of multi-block and domain decomposition-based problems, which form an important kind of problem in the high-performance computing area.
- The *farm* skeleton improves a task throughput as different data sets can be computed in parallel on different sets of processors.

- Problem solutions that have a communication pattern based on array interchange can be defined and solved easily by using the *pipeline* skeleton, which pipelines sequences of tasks.

Scientific component interfaces are described by means of two different views. The *application view* contains information related to data types of component input/output. This view is used by the programmer in order to develop his/her applications by means of the composition language. The *configuration view* extends the application view with information about input and output data distribution, processor layout and component internal structure (in terms of the skeleton composition scheme).

The knowledge at the component interface level of data distribution and processor layout allows the system to obtain an efficient implementation of the communication scheme among components, which follows a "data flow" style by means of a typical `put_data/get_data` scheme.

The implementation of SBASCO is based on the extension of the message passing interface, namely MPI-2 [3]. The system exploits the functions for the creation and management of processes, as well as the mechanisms for connecting and communicating parallel MPI applications. The component model is supported by a compiler and libraries to facilitate the programming task.

This section has summarized the main features of SBASCO. A more detailed explanation can be found in [2].

3 Problem Formulation and Discretization

Physical phenomena involving heat and mass transfer, combustion, etc. are characterized by reaction-diffusion equations with non-linear source terms. Here, we consider the following set of two time-dependent, nonlinearly coupled PDEs:

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + S(U), \tag{1}$$

where

$$U = (u, v)^T, \qquad S = (-uv, uv - \lambda v)^T, \tag{2}$$

u and v represent the concentration of a reactant and the temperature, respectively, $u = 1$ and $v = 0$ on the external boundaries, t is time, x and y denote Cartesian coordinates, λ is a constant (in this paper, $\lambda = 0.5$), and the superscript T denotes transpose. Eq. (1) has been previously studied in [6], where a comparison of several numerical techniques for tackling domain decomposition problems in irregular domains is presented.

Eq. (1) was discretized by means of an implicit, linearized, θ-method in an equally spaced grid where the non-linear term $S_{i,j}^{n+1}$ was approximated by means of its Taylor polynomial of first degree around (t^n, x_i, y_j) to obtain the following system of linear algebraic equations:

$$\begin{aligned}\frac{\Delta U_{i,j}}{k} = {} & \frac{1}{\Delta x^2}\left[\theta\delta_x^2 \Delta U_{i,j} + \delta_x^2 U_{i,j}^n\right] + \frac{1}{\Delta y^2}\left[\theta\delta_y^2 \Delta U_{i,j} + \delta_y^2 U_{i,j}^n\right] \\ & + S_{i,j}^n + \theta J_{i,j}^n \Delta U_{i,j},\end{aligned} \tag{3}$$

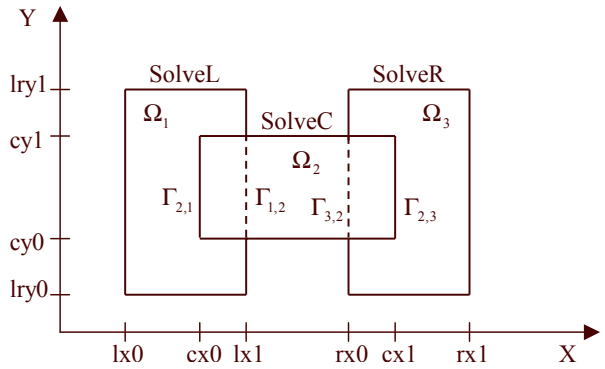

Fig. 1. Original domain decomposed into overlapping regions

where

$$\Delta U_{i,j} = U_{i,j}^{n+1} - U_{i,j}^{n}, \quad S_{i,j}^{n} = S(U_{i,j}^{n}), \quad J_{i,j}^{n} = \frac{\partial S}{\partial U}(t^n, x_i, y_j)$$
$$\delta_x^2 U_{i,j} = U_{i+1,j} - 2U_{i,j} + U_{i-1,j}, \quad \delta_y^2 U_{i,j} = U_{i,j+1} - 2U_{i,j} + U_{i,j-1}, \quad (4)$$

i and j denote x_i and y_j, respectively, t^n denotes the nth time level, k is the time step, Δx and Δy represent the grid spacing in the $x-$ and $y-$directions, respectively, and $0 < \theta \leq 1$ is the implicitness parameter. In this paper, $\theta = 0.5$, i.e., second-order accurate finite difference methods are employed.

4 Application Design and Implementation

In order to apply domain decomposition, the geometry of the problem being considered is divided, as shown in Fig. 1, into several overlapping regions, i.e. Ω_1, Ω_2 and Ω_3. The part of the boundary of Ω_i that is interior to Ω_j is denoted by $\Gamma_{i,j}$. In the Dirichlet method considered here, the current solution in one subdomain is used to update the boundary data in the adjacent subdomains. Then, the interior points are recalculated by solving the corresponding linear system of algebraic equations. This iterative procedure is repeated until convergence. The following algorithm gives an abstract description of the global application.

```
1.  Set initial conditions                  (on each subdomain)
2.  For time_step = 1..MAX_TIME_STEPS Do
3.    Repeat
4.      Update boundaries                   (on each subdomain)
5.      Solve system of linear equations    (on each subdomain)
6.    Until convergence on ALL subdomains
```

This algorithm involves up to three major nested loops: the `For` loop in line 2, the `Repeat` loop in line 3 and the linear system solver in line 5 which, due

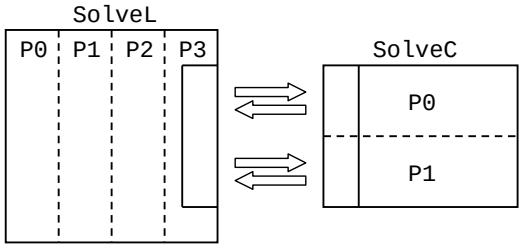

Fig. 2. Boundaries updating involves parallel communications

to the large number of unknowns per domain, is usually based on an iterative procedure, instead of a direct method. Regarding the convergence criterion in line 6, this is satisfied when boundary values are not significantly modified in two iterations ($\|\Gamma_{i,j}^{k} - \Gamma_{i,j}^{k-1}\| < 10^{-10}$ for every pair of adjacent subdomains).

A way to execute the above algorithm in parallel is to associate a task with each one of the regions. In our approach, these tasks are encapsulated into software components which comprise the application so that, we have instances of components that run in parallel to solve the problem. Furthermore, the scientific code implemented in components can be sequential or parallel as well. For example, we use parallel red-black Gauss-Seidel relaxation as a linear equations solver and so, data parallel components are considered.

The updating of the boundaries and the convergence criterion (lines 4 and 6 respectively) require the synchronization and communication of components. Data distribution and processor layout are the key elements that influence the implementation of an efficient communication scheme, as is shown in Fig. 2. The component on the left, called `SolveL`, is executed on four processors its domain being distributed by columns. The component on the right, called `SolveC`, runs on two processors having data distributed by rows. When boundaries are being updated, processor `P3` of `SolveL` needs to communicate with processors `P0` and `P1` of `SolveC`. It is important to remark that the system manages these communications automatically, according to the high-level description of boundaries, data distribution and processor layout that the programmer has provided.

The SBASCO solution uses a *multiblock* skeleton, which is oriented to these types of domain-decomposition problems, in order to structure the application and establish the interaction scheme among the components.

```
1.  PROGRAM ReactionDiffusion
2.  complex, DOMAIN2D :: left/lx0,lry0,lx1,lry1/,
3.                       center/cx0,cy0,cx1,cy1/,
4.                       right/rx0,lry0,rx1,lry1/
5.  STRUCTURE
6.    MULTIBLOCK
7.      SolveL(left) ON PROCS(4),
8.      SolveC(center) ON PROCS(2),
9.      SolveR(right) ON PROCS(4)
```

```
10.   WITH BORDERS
11.     left(lx1,cy0,lx1,cy1)<-center(_),
12.     center(cx0,cy0,cx0,cy1)<-left(_),
13.     /* ... */
```

The program declares three domains based on the geometry shown in Fig. 1. The `complex` datatype was chosen because every domain point needs to save a pair of values, i.e. u and v. The `MULTIBLOCK` definition includes the selected components, the number of processors assigned to each one, and the boundaries definition. The expression `left(lx1,cy0,lx1,cy1)<-center(_)` indicates that the zone of `left` delimited by points `(lx1,cy0)` and `(lx1,cy1)` will be updated (in every iteration) by the values belonging to the zone of `center` delimited by the same points. No data distribution is declared at the composition level, as this information will be obtained from the configuration view of the different components involved. Distribution types are similar to those used in HPF. The following shows the configuration view of `SolveC`:

```
1.  CONFIGURATION INTERFACE SolveC
2.    complex, INOUT, DOMAIN2D :: my_domain,
3.    DISTRIBUTE my_domain(BLOCK,*)
```

By using the high-level *mutiblock* skeleton, the programmer is abstracted from details such as the synchronization and communication among parallel components or the execution pattern implemented in the domain decomposition method. So, he/she is only concerned with the programming of the initial conditions, the desired convergence criterion and the linear system solver. As a result, the complexity of the application development decreases significantly.

The programming of the components is carried out in an object-oriented style using C++ with MPI, together with a class library provided by SBASCO. The software architecture is comprised of a group of MPI applications that implements the parallel components being executed on disjoint sets of processors. The creation and interconnection of scientific components are carried out by means of the process management mechanisms defined in MPI-2. Prior to the application execution, the components exchange information related to data distribution of arguments and processor layout so that message passing can be performed efficiently when communication takes place.

5 Experimental Results

The presented application has been used to solve Eq.(1) on a domain with reentrant corners, which is decomposed into three overlapping regions as described in Fig. 1. The problem is subjected to the following initial conditions:

$$u(x, y, 0) = 1, \quad v(x, y, 0) = e^{-((x-x_{ig})^2+(y-y_{ig})^2)}, \tag{5}$$

where (x_{ig}, y_{ig}) is located at the center of Ω_1 and denotes the ignition location. These conditions establish a peak of temperature in the left subdomain needed to start the reaction process.

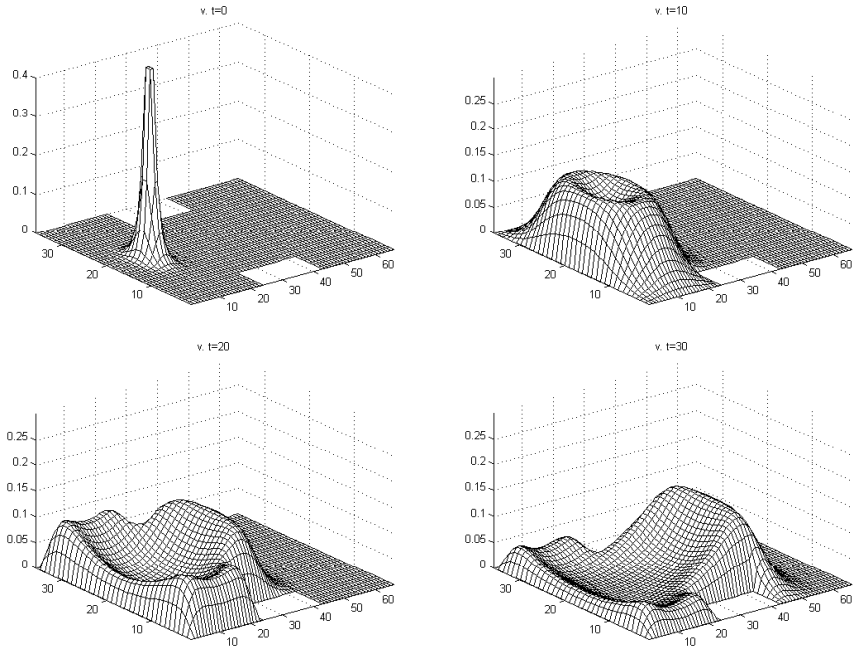

Fig. 3. Spatial distribution of nondimensional temperature, v, at $t = 0, 10, 20$ and 30

The solution is characterized by a reaction front that propagates from the ignition point to the rest of the domain. The concentration of reactant, u, decreases as the reaction progresses, whereas the temperature, v, exhibits the behavior shown in Fig. 3.

Table 1 summarizes the computational time, in seconds, for different combinations of grid sizes and processors. Each problem instance is composed of three same-sized subdomains which are overlapped. Column one denotes the size of the problem as the area of the smallest rectangle that contains the complete domain. The number of processors is equally distributed among the three subdomains. In addition, we provide a sequential case in which all components are executed on a single processor. Experiments were carried out in a cluster of Pentium 4, 2.66GHz, 1GB RAM Linux workstations interconnected with a 1Gb/s Myrinet network and running LAM/MPI as message passing interface implementation. As the results show, the high-level programming approach used does not result in loss of performance when the number of processors is increased.

6 Conclusions

This paper has shown a practical utilization of modern high-level programming techniques in scientific computing. The presented application solves a system of two reaction-diffusion equations by means of domain decomposition methods on

Table 1. Execution time, in seconds, for the reaction-diffusion problem. Values in brackets represent speed-up.

Problem size	Subdomain size	Sequential	Number of processors			
			3	6	9	12
240x440	240x160	54.45	30.22	19.72	14.71	13.12
			(1.80)	(2.76)	(3.70)	(4.15)
480x880	480x320	189.62	109.54	65.80	48.67	42.78
			(1.73)	(2.88)	(3.89)	(4.43)
960x1760	960x640	805.23	548.18	329.16	195.09	173.44
			(1.46)	(2.44)	(4.12)	(4.64)

irregular domains that exhibit re-entrant corners. Based on the high-level parallel programming environment SBASCO, the approach takes advantage of using algorithmic skeletons and software components, two technologies that elevate the abstraction degree of the programming model, as a consequence, making the development of the parallel application easier, as well as improving the software evolution and maintenance. Some experimental results have shown the suitability and efficiency of the proposal.

References

1. Cole, M., Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, 1989.
2. Díaz, M., Rubio, B., Soler, E., Troya, J.M., SBASCO: Skeleton-Based Scientific Components, in "Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2004)", pp. 318–324, IEEE Computer Society, A Coruña, Spain, 2004.
3. Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., Snir, M., MPI: The Complete Reference, volume 2–The MPI-2 Extensions. MIT Press, 1998.
4. Heineman, G.T., Council, W.T., Component-Based Software Engineering: Putting the Pieces Together. Addision Wesley, 2001.
5. Lions, P.L., On the Schwarz Alternating Method III, in "Proceedings of the 3rd International Symposium on Domain Decomposition Methods for Partial Differential Equations", pp. 202–223, Chain et al. eds, Philadelphia, SIAM, USA, 1989.
6. Ramos, J.I., Soler, E., Domain Decomposition Techniques for Reaction-Diffusion Equations in Two-Dimensional Regions with Re-entrant Corners, *Applied Mathematics and Computation*, **118**, 2-3 (2001), pp. 189–221.
7. Smith, B., Bjørstad, P., Gropp, W., Domain Decomposition. Parallel Multilevel Methods for Elliptic P.D.E.'e. Cambridge University Press, 1996.
8. Quarteroni, A., Valli, A., Domain Decomposition for Partial Differential Equations. Oxford Science Publications, 1999.