Dynamic Reconfiguration of Scientific Components using Aspect Oriented Programming: A Case Study

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. This paper is a case study on the use of a high-level, aspect-oriented programming technology for the modelling of the communication and interaction scheme that affects the set of components of a parallel scientific application. Such application uses domain decomposition methods and static grid adaptation techniques in order to obtain the numerical solution of a reaction-diffusion problem modelled as a system of two time dependent, non-linearly coupled partial differential equations. Besides achieving the usual advantages in terms of modularity and reusability, we pursue to improve the efficiency by means of dynamic changes of aspects at runtime. The application design considers two types of components as first-order entities: Scientific Components (SCs) for the computational tasks and Communication Aspect Components (CACs) for the dynamic management of the communication among SCs. The experiments show the suitability of the proposal as well as the performance benefits obtained by readjusting the communication aspect.

1 Introduction

Recently, significant efforts are being made in order to incorporate component-oriented programming [5] to develop new parallel and distributed programming environments (PEs) in the high-performance computing area. In this sense, ASSIST [13] is focused on high-level programmability and software productivity for complex multidisciplinary applications, including data-intensive and interactive software. SBASCO [1] is a PE oriented to the efficient development of parallel and distributed numerical applications. A large effort is currently devoted to define a standard component architecture for high-performance computing in the context of the Common Component Architecture (CCA) Forum [12].

On the other hand, aspect-oriented programming (AOP) [6] enables developers to capture crosscutting structure that is spread over the different components of a system. This allows to program crosscutting concerns in a modular way, and achieve the usual benefits of improved modularity: simpler code that is easier to develop and maintain, and that has a greater potential for reuse. A well-modularised crosscutting concern is called an aspect.

This paper discusses the development of a parallel scientific application using a high-level programming technology that integrates software components and aspects. The solution is based on the SBASCO extension described in [2], which considers the encapsulation of several functionalities affecting the set of scientific components into separated aspects which are also modelled as components. More specifically, this work is focused on the dynamic management of the application communication scheme and processor layout.

The reaction-diffusion problem considered here is described as a system of two time-dependant, non-linearly coupled partial differential equations (PDEs). Domain decomposition methods [9] are used to achieve the solution in a two-dimensional rectangular domain of long-aspect ratio, i.e. $L_x > L_y$. These methods divide the original geometry into separated regions so that a parallelization strategy can be applied for tackling large-scale and realistic problems [11].

The problem solution is characterized by the presence of a travelling wave front in the x-direction. Due to the steepness and the curvature of the front, a large number of grid points are needed to accurately calculate the numerical solution of the system of equations. Alternatives to equally spaced grids that can reduce the computational cost are those based on grid-adaptive procedures [7]. We use a static grid adaption technique that concentrates the grid points in an adaptive static fashion on regions where they are needed.

Two different parallelism levels are combined in the application. On the one hand, the solution in the different subdomains is computed in parallel by assigning one task (component) per subdomain. On the other hand, scientific components use data-parallel red-black Gauss-Seidel relaxation to solve the large system of equations so that data-parallel components are considered.

The high-level application design presented allows the developer to be released from the tedious and error-prone parallelism exploitation details, such as the creation, synchronization and communication of data-parallel tasks, which are automatically managed by the runtime support. The communication scheme is encapsulated into a communication aspect component (CAC) to which the different scientific components (SCs) are connected. During the execution, the CAC can detect that a SC is running slower than the others, which decreases the application performance. In response, the CAC can dynamically carry out a re-mapping of the processors being assigned to the SCs and establish a new efficient communication scheme for component interaction.

This paper is structured as follows. The next section outlines the fundamentals of the framework the solution is based on. Section 3 introduces the physical problem taken into consideration. Section 4 is devoted to describe in detail the application design and implementation. Experimental results are shown in section 5. Finally, some conclusions are outlined.

2 Aspect-Oriented Framework Overview

The aspect-oriented framework [2] used in this work means an evolution of SBASCO in the sense that the former extends the latter in an attempt to apply

the AOP paradigm to the high-performance computing scene. The application domain is the parallel and distributed solution of numerical problems, especially those with an irregular surface that can be decomposed into regular, block structured domains, and other kind of problems that take advantage of integrating task and data parallelism and have a communication pattern based on (sub)array interchange. Usually, their execution times are large.

The approach provides the basis to separate specific aspects that appear in the above-mentioned application domain, such as communication, synchronization, distribution, convergence criteria, numerical method, boundary conditions, etc, which are managed as first-order entities.

There are two types of components: scientific components (SCs), which are in charge of computational tasks, and Aspect Components (ACs) used to encapsulate crosscutting concerns. Both types of components are combined in order to construct "aspectized" scientific applications.

SCs interact each other following a "data flow" style by means of a typical `get_data/put_data` scheme based on the SBASCO approach, whereas the interaction among SCs and ACs follows a "procedural" style based on asynchronous method calls. The language used to specify the interface of both SCs and ACs has a syntax similar to that used in CCM [8], enriched by constructors and data types characterizing the programming model of SBASCO.

In SBASCO, applications are static in the sense that configuration parameters such as the number of processors assigned to each component are fixed at the composition level. However, the evolution of the system during the execution may require a dynamic readjustment of these parameters in order to preserve the efficiency. As a consequence, the communication scheme among the different components, which depends on this information, should also be modified.

In our approach, the communication scheme is an example of crosscutting concern which is encapsulated into a communication aspect component (CAC). Such scheme is described using a software skeleton which specify data distribution, processor layout and, possibly, number of replicas of each SC. The following types of skeletons are inherited from the SBASCO programming model:

- 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.

The strategy or actions followed to dynamically reconfigure an application as well as the conditions under which such actions are applied can be easily programmed as part of the CAC.

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 [10], 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 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:

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

where

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

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

This section covers a series of topics such as the algorithm description and its parallelization, the application design and the programming of the components.

4.1 Algorithm Description and Parallelization Strategy

In order to apply domain decomposition, the geometry of the problem is divided into several overlapping subdomains. In the example shown in Fig. 1, each subdomain $\Omega_k, 1 \leq k \leq 4$, is the rectangular area delimited by the cartesian points

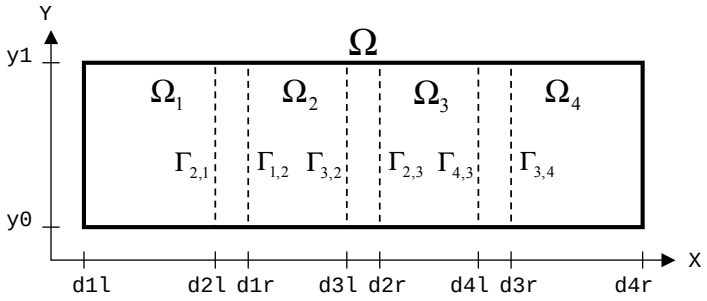

Fig. 1. Original two-dimensional domain divided into overlapping regions

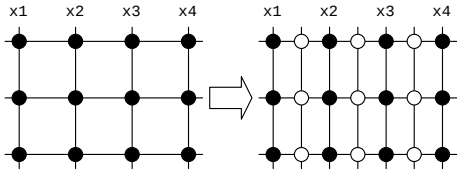

Fig. 2. Double-sized grid used to improve the accurate of the solution

(dkl,y0) and (dkr,y1). 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 system of linear equations, Eq. (3). This iterative procedure is repeated until convergence.

In addition, we are interested in obtaining a more accurate numerical result in those areas where the solution presents steep gradients and high curvature. A simple static grid refinement procedure is used to increase the number of grid points where the travelling wave front is located. As the propagation of the front is mainly carried out in the x-direction, the number of points in that direction is duplicated as well as the grid spacing parameter, Δx, is reduced by half. The unknown points of the refined grid, which are denoted by white points in Fig. 2, are calculated by means of cubic spline interpolation using the current solution available in the black points. This procedure is applied when $|[\partial v/\partial x]|$ exceeds a predefined constant. From that instant, the double-sized grid is used in the calculation of each time step until the previous criterion fails, which means that the wave front is not significantly affecting the subdomain.

The operations carried out in each subdomain are summarized as follows:

```
1.  Set initial conditions; is_duplicated = FALSE
2.  For time_step = 1..MAX_TIME_STEPS Do
3.    If (is_duplicated = FALSE) AND (slope > eps) Then
4.      Interpolate; use double-sized grid; is_duplicated = TRUE
5.    If (is_duplicated = TRUE) AND (slope < eps) Then
6.      Use simple grid; is_duplicated = FALSE
```

```
Interface ScConfigure {
  void go(in Domain2D dom, in CommScheme scheme, in unsigned procs,
          in unsigned iters);
  void continue(in unsigned iters);
  void changeCommScheme(in CommScheme scheme);
  void changeNumProcs(in unsigned procs);
};
Interface CacProvided {
  void finished(in SC sc);
};

component SC {
  provides ScConfigure conf;
  uses CacProvided cac;
};
component Solve1 : SC{};
component Solve2 : SC{};
...

component CAC {
  skeleton ReactionDiffusion {
    Domain2D dom1 = {d1l,y0,d1r,y1},
             dom2 = {d2l,y0,d2r,y1},
             ...
    Multiblock {
      Solve1(dom1:(BLOCK,*)) ON PROCS(2);
      Solve2(dom2:(BLOCK,*)) ON PROCS(2);
      ...
    WithBorders
      dom1(d1r,y0,d1r,y1) <- dom2(_);
      dom2(d2l,y0,d2l,y1) <- dom1(_);
      ...
    };
  };
  provides CacProvided cac;
  uses ScConfigure conf;
};
```

Fig. 3. Definition of SCs and CAC for the reaction-diffusion application

```
7.    Repeat
8.      Update boundaries
9.      Solve system of linear equations
10.   Until convergence on ALL subdomains
```

The above algorithm is parallelized by associating one task per subdomain. These tasks are encapsulated into scientific components composing the application so that, we have different instances of components running in parallel to solve the problem. In addition, components internally exploit parallelism as they use data-parallel red-black Gauss-Seidel relaxation to solve the system of linear equations. As a consequence, this application design means the efficient integration of both data and task parallelism.

4.2 Scientific and Communication Aspect Components

The software components considered in the application development are shown in Fig. 3. The definition of the CAC, besides specifying the provided and used interfaces, establishes the `Multiblock` skeleton that describes the interaction scheme among the SCs. The predefined type `Domain2D` is used to capture the Cartesian points that establish the region of a domain [3].

```
void Cac::configRule() {
  int i;
  Component *comp, *worstComp;

  worstComp = appComponents.worst;
  if (appComponents.availableProcs > 0)
    worstComp->increaseProcs(1);
  else {
    for (i=0; i < appComponents.numComps; i++)
      if (appComponents.fromBestToWorst[i]->procs > 1) {
        comp = appComponents.fromBestToWorst[i];
        break;
      }
    if ((i < appComponents.numComps) &&
        (comp->execTime < worstComp->execTime * alpha)) {
      comp->decreaseProcs(1);
      worstComp->increaseProcs(1);
    }
  }
}
```

Fig. 4. Implementation of a re-configuration strategy in the CAC

On the other hand, the type `CommScheme` will contain the required information, which is declared in the skeleton, in order to carry out an efficient inter-component communication: data distribution, domain interactions and mapping of processors. The data distribution types are HPF-like. For example, the expression `dom1:(BLOCK,*)` declares a distribution by rows for the domain `dom1`.

As stated before, the problem is solved iteratively, so that for each iteration a communication of the internal boundaries, which are defined among the domains, is needed. For example, the expression `dom1(d1r,y0,d1r,y1)<-dom2(_)` indicates that the zone of `dom1` delimited by points `(d1r,y0)` and `(d1r,y1)` will be updated by the values belonging to the zone of `dom2` delimited by the same points. SCs communicate their internal boundaries by means of the predefined data flow primitives `get_data` and `put_data` which, respectively, obtain data from and send data to a SC.

In addition, the relationship between domains and the processor layout where the SCs are going to be executed are also indicated, e.g. `ON PROCS(2)`. Each SC is solved by a disjoint set of processors.

4.3 Application Programming and Execution

The development of both scientific and aspect components is carried out following an object-oriented programming style based on C++ and MPI.

The programming of the CAC includes the implementation of two methods in the class that represents the component in order to allow the programming of the dynamic reconfiguration procedure in an easy way.

The `history` method uses the predefined object `schedule` to establish a set of time stamps that represent the points at which dynamic changes in the application will be (possibly) applied. For example a call to `schedule.cycle(100)` means each time a hundred of time steps are computed by the SCs, the CAC

decides if a re-configuration of the application is needed. Time stamps can also be individually set up by calling `schedule.setTimeStamp( time_stamp)`.

The reconfiguration operation is coded in the `configRule` method. The programmer uses the `appComponents` object to query the different components and perform actions on them. Figure 4 proposes a very simple but effective implementation of this method. It must be noted that much more complicated strategies based on the evaluation of a large amount of parameters could be applied.

The programming of the SCs consists in writing the scientific code. The problem initial conditions are established in the `initialize` method. The code of the `task` method is executed each time new input data are available. Most of the functions are called from here: the selection between the simple and double-sized grid, the interpolation procedure, the method to solve the system of equations, etc. The data-parallel Gauss-Seidel technique is implemented using MPI.

When the number of processors of a SC is dynamically changed, the system guarantees that the current state of the computation is maintained by means of a correct data re-distribution. This is automatically carried out for the domain, which is the only data declared at the composition level. However, other internal variables may also be considered as part of the component state. For that reason, the SC provides the methods `preMapping` and `postMapping` which are executed, respectively, just before and after the re-mapping of processors. The programmer can implement these operations to communicate his/her variables as needed.

The sequence of application execution begins when the CAC calls `go` on the SCs, so that the domain, initial communication scheme and number of processors are established. The number of iterations used corresponds to the first time stamp. The `go` operation calls `initialize` and `task`. When a SC finishes its computational task, it notifies the CAC calling `finished`. Then, the `configRule` method evaluates the reconfiguration of the application. For example, let us suppose that the number of processors of the SC `Solve1` needs to be incremented by one. In that case, the CAC calls `changeNumProcs` on such SC, which executes the redistribution code. Additionally, some other SCs must be informed about the change in the communication scheme. More specifically, the CAC calls `changeCommScheme` on `Solve2` as it is the only component that communicates with `Solve1` (subdomains `dom1` and `dom2` are connected by internal boundaries, see Fig 3). The rest of components can preserve their communication scheme. Finally, the CAC calls `continue` on the SCs so that the computation goes on.

The solution is constructed in the context of the framework introduced in section 2. The reconfiguration of SCs is based on the process management functions defined in MPI-2 [4]. The standard also provides the mechanisms for communicating stand-alone parallel applications (SCs in our context).

5 Experimental Results

The application has been used to solve Eq. (1) in a rectangular domain using four overlapping subdomains as described in Fig. 1.

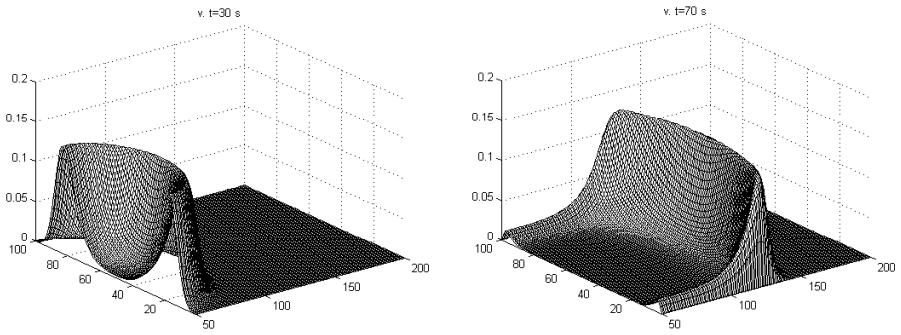

Fig. 5. Spatial distribution of temperature at $t = 30$ (left), and 70 seconds (right)

Table 1. Static and dynamic configuration for the reaction-diffusion problem. Execution time is expressed in seconds. Values in brackets mean improvement percentage

		Number of processors					
Domain	Type	4	6	8	10	12	14
100x340	static	1236.4	1045.2	719.7	641.1	505.5	479.3
	dynamic	-	655.2	532.9	473.4	415.7	379.9
		-	(37.31%)	(25.95%)	(26.15%)	(17.76%)	(20.73%)
100x740	static	4656.8	3645.6	2665.5	2268.6	1884.1	1767.83
	dynamic	-	2264.7	1938.6	1559.0	1358.6	1236.05
		-	(37.87%)	(27.27%)	(31.27%)	(27.88%)	(30.08%)

The solution is characterized by a reaction front that propagates across the domain. Once the front reaches the upper and lower borders, the propagation is carried out in the x-direction, as shown in Fig. 5.

Table 1 shows the execution time, in seconds, for the different experiments carried out. Two problem instances have been considered, based on the use of subdomains of sizes 100x100 and 100x200, respectively, with an overlapping length of 20 points. Processors are equally distributed among the SCs. In cases where the number of processors is not a multiple of 4, the two SCs associated with the domains Ω_2 and Ω_3 (see Fig. 1) obtain one additional processor each. Two types of component configurations are evaluated: static, using a fixed processor mapping, and dynamic, carrying out processor re-assignments at runtime based on the strategy described in Fig. 4.

Results show significant improvements when using dynamic reconfiguration of components. In our application, the computational cost of a scientific component evolves as it is mainly influenced by the grid refinement technique used. The synchronization of tasks imposed by the boundaries updating means that

the overall performance of the application decreases when any of the components is running slower than the others. In such conditions, any of the results obtained by means of a static processor mapping will incur in a performance reduction as some of the processors will be idle during some periods of time. The redistribution of processors at runtime overcomes this shortcoming.

6 Conclusions

This paper addresses the development of a parallel scientific application in order to resolve a system of time-dependent and nonlinear PDEs using domain decomposition methods and static grid adaptation procedures.

The high-level approach on which the application is based considers scientific components (SCs) for the encapsulation of the data-parallel tasks. SCs interact each other following a communication scheme that depends on factors such as data distribution and processor layout. Such communication scheme, which represents a crosscutting concern in our system, is captured and managed in a communication aspect component (CAC).

The CAC makes dynamic readjustments in the processor mapping and informs the SCs about the changes in the communication scheme so that component interactions are still being carried out efficiently. As a result, the application performance is improved.

References

1. 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.
2. Díaz, M., Romero, S., Rubio, B., Soler, E., Troya, J.M., An Aspect-Oriented Framework for Scientific Component Development, in "Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2005)", pp. 290–296, IEEE Computer Society, Lugano, Switzerland, 2005.
3. Díaz, M., Rubio, B., Soler, E., Troya, J.M., A Border-based Coordination Language for Integrating Task and Data Parallelism, *Journal of Parallel and Distributed Computing*, **62**, 4, pp. 715–740, 2002.
4. 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.
5. Heineman, G.T., Council, W.T., Component-Based Software Engineering: Putting the Pieces Together. Addision Wesley, 2001.
6. Kiczales, G., et. al., Aspect-Oriented Programming, in "Proceedings of the Europe Conference on Object-Oriented Programming (ECOOP 1997)", LNCS 1241, pp. 220–242, Springer, Jyvskyl, Finland, 1997.
7. Lang, J., High-Resolution Self-Adaptive Methods for Turbulent Chemically Reacting Flows, *Chemical Engineering Science*, **51**, pp. 1055–1070, 1996.

8. Object Management Group (OMG), Specification of Corba Component Model (CCM), home page:
http://www.omg.org/technology/documents/formal/components.htm.
9. Quarteroni, A., Valli, A., Domain Decomposition for Partial Differential Equations. Oxford Science Publications, 1999.
10. 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.
11. Smith, B., Bjørstad, P., Gropp, W., Domain Decomposition. Parallel Multilevel Methods for Elliptic P.D.E.'e. Cambridge University Press, 1996.
12. The Common Component Architecture Forum, home page:
http://www.cca-forum.org.
13. Vanneschi, M., The Programming Model of ASSIST, an Environment for Parallel and Distributed Portable Applications, *Parallel Computing*, **28**, 12, pp. 1709–1732, 2002.