Integrating Multiple Implementations and Structure Exploitation in the Component-based Design of Parallel ODE Solvers

J. M. Mantas[1], J. Ortega Lopera[2], and J. A. Carrillo[3]

[1] Software Engineering Department. University of Granada
C/ P. Daniel de Saucedo s/n. 18071 Granada, Spain. jmmantas@ugr.es
[2] Computer Architecture and Technology Department. University of Granada.
C/ P. Daniel de Saucedo s/n. 18071 Granada, Spain. jortega@atc.ugr.es
[3] Departament de Matematiques - ICREA. Universitat Autonoma de Barcelona
Bellaterra E-08193. carrillo@mat.uab.es

Abstract. A COmponent-based Methodology to derive Parallel programs to solve Ordinary Differential Equation (ODE) Solvers, termed COMPODES, is presented. The approach is useful to obtain distributed implementations of numerical algorithms which can be specified by combining linear algebra operations. The main contribution of the approach is the possibility of managing several implementations of the operations and exploiting the problem structure in an elegant and systematic way. As a result, software reusability is enhanced and a clear structuring of the derivation process is achieved. The approach includes a technique to take the lowest level decisions systematically and is illustrated by deriving several implementations of a numerical scheme to solve stiff ODEs.

1 Introduction

One important formulation for the ODEs arising from the modelling process is that of the *Initial Value Problem* (IVP) for ODE [3]. The goal in the IVP is to find a function $y : \mathbb{R} \rightarrow \mathbb{R}^d$ at an interval $[t_0, t_f]$ given its value at t_0, $y(t_0) = y_0$, and a system function $f : \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}^d$ fulfilling that $y^{'}(t) = f(t, y)$.

The computational demands of the numerical methods to solve IVPs, and the complexity of the IVPs which arise in practical applications, suggests the use of efficient algorithms for Distributed-Memory Parallel Machines (DMPMs).

In order to achieve an acceptable performance on a DMPM, the parallel design of this kind of numerical software must take into account 1) the task and data parallelism exhibited by the method (usually, several subcomputations in a time step can be executed simultaneously and each one is composed of linear algebra operations) 2) the characteristics of the particular DMPM and 3) the particular structure of the IVP whose exploitation is fundamental.

Currently, the development of parallel software for these applications benefits from two contributions:

• The development of reusable software components of parallel linear algebra libraries [2] for DMPMs, which encapsulate the details about the efficient SPMD implementation of many standard solution methods.

• The hierarchical execution systems to exploit the mixed parallelism [7, 8] facilitate the generation of group SPMD programs in which several SPMD subprograms are executed by disjoint processor groups in parallel. This execution model is specially suitable to exploit the potential task and data parallelism of these applications and the MPI standard makes it possible its implementation on DMPMs. The *TwoL* framework [8] for the derivation of parallel programs and the PARADIGM compiler system [7] are relevant contributions in this area. However, these approaches may benefit from explicit constructs to: a) maintain and select among multiple implementations of an operation in an elegant and systematic way (called *performance polymorphism* in [5, 4]), and b) exploit the particular structure of the IVPs.

We propose a methodological approach based on linear algebra components for deriving group SPMD programs to solve ODEs. This approach, termed COMPODES, includes explicit constructs for performance polymorphism, takes into account the exploitation of the problem structure and enables the structuring of the derivation process by including three clearly defined phases in which optimizations of different types can be carried out. The proposal is illustrated by deriving parallel implementations, adapted to two different IVPs, of an advanced numerical scheme [9, 4] to solve stiff ODEs.

A brief introduction to COMPODES is presented in section 2. The three following sections illustrate the COMPODES phases and introduce a technique to complete the last phase. Some conclusions are drawn in section 6.

2 COMPODES Overview

In COMPODES, the description of the functional behaviour of a linear algebra operation is decoupled from the multiple implementations which provide the same functionality. This is achieved by encapsulating each entity as separate software components [5, 4]: *concepts* (formal definitions of operations) and *realizations* (particular implementations). Each concept can have several associated realizations. This explicit distinction allows us to select the implementation that offers the best performance in a particular context (target machine, IVP, etc.).

A concept (`MJacobian`) that denotes the computation of an approximation to the Jacobian of a function f at a point $(t, y) \in \mathbb{R}^{d+1}$ is presented in Figure 1. Two different realizations for `MJacobian` are shown: $Seq\_MJacobian$ encapsulates a sequential implementation of the operation and the parallel realization $Block\_MJacobian$ obtains a block column distribution of the Jacobian matrix.

A realization has a client-visible part, called the *header*, which describes the aspects that must be known to use the code (called the *body*) including: a) the distribution of the array arguments among the processors [4], b) a formula which estimates its runtime from parameters describing the data distributions, the target machine (number of processors P, per word communication cost t_w, startup time t_s, etc.), and problem size, and c) the storage scheme of the arguments.

We have defined a specialization mechanism to adapt a concept to deal with the particular structure of its arguments. When a concept has not been obtained by the specialization of a previously existing one, we say that it is a *general*

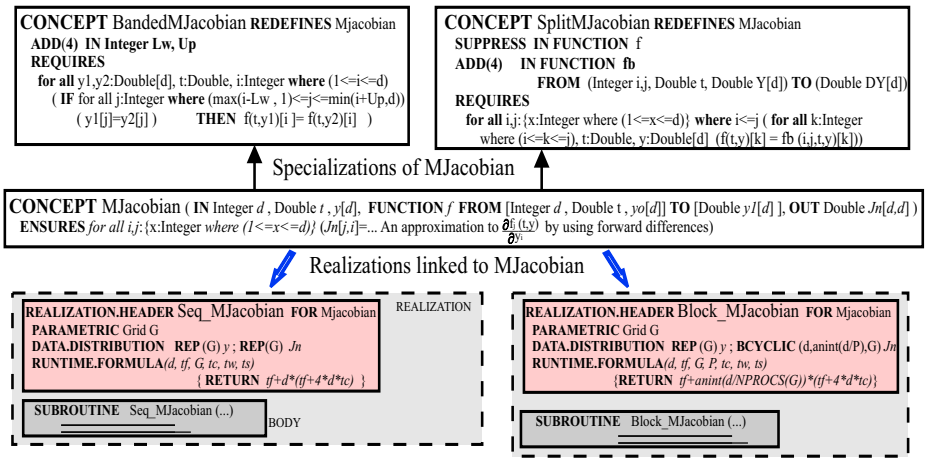


Fig. 1. The `MJacobian` concept with several realizations and specializations

concept. A specialized concept might admit more efficient implementations than a general one because the implementation can take advantage of the special structure of its arguments. We can consider two types of specialization:

• Specializations based on the array structure [5]. For instance, we can specialize a concept which denotes the usual matrix-vector product operation (`MVproduct`) to deal with banded matrices (`BandedMVproduct`).

• Specializations based on the function structure. In Figure 1, several specializations of the `MJacobian` concept based on the function structure are shown. `BandedMJacobian` specializes `MJacobian` to deal with a banded system function with bandwith $Lw + Up + 1$. A system function f is banded when a component of the output vector $f(t, y)$ only depends on a consecutive band of the input vector y. This kind of function enables the minimization of the remote communication when the function is evaluated on several processors. The Jacobian matrix generated with `Banded_MJacobian` must also be banded. The concept `SplitMJacobian` indicates that the argument f is cost-separable. A system function is *cost-separable* when it is possible to define a *partitioning function* fb which allows the evaluation of f homogeneously by pieces without introducing redundant computation. This property is important to enable the easy exploitation of the data parallelism in the block parallel evaluation of f.

The derivation process in COMPODES starts from a mathematical description of the numerical method to solve ODEs and proceeds through several phases. During the first phase, called **functional composition**, several general concepts are selected and combined by using constructors of sequential and concurrent composition to describe the functionality of the method and the maximum degree of task parallelism. As a result, a version of *general functional specification* is obtained (see Figure 2b). From this representation, one can obtain a sequential program automatically (by using the sequential realizations linked to the general concepts which appear in the specification), which makes it possible to validate the numerical properties of the method. If the validation detects neither anoma-

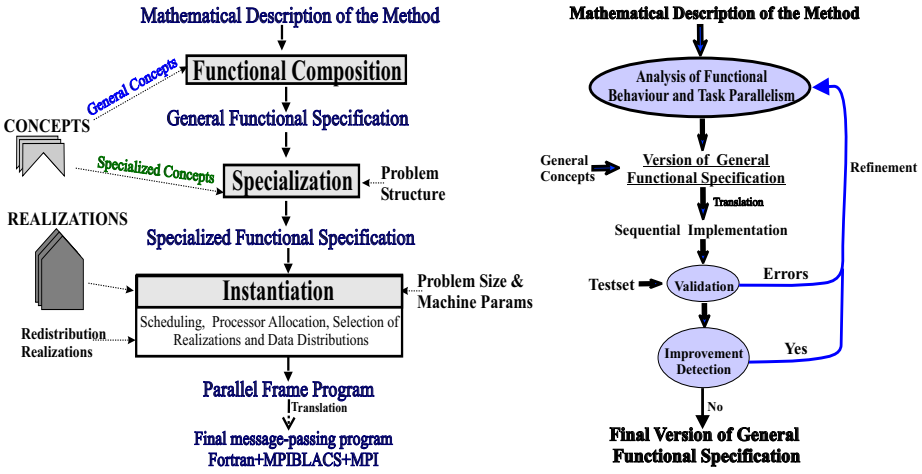


Fig. 2. a) Overview of COMPODES, b) Funtional Composition

lies nor improvement possibilities, we obtain a definitive version of the general specification which can be used in the solution in different contexts of different IVPs on different architectures.

During the **specialization** phase, the general specification is modified according to the structural characteristics of the particular IVP to be solved. These modifications consist fundamentally of replacing general concepts by specializations. The user must provide a sequential routine to evaluate the system function and a sequential prototype can also be generated to validate the specification. As a result, a *specialized specification* is obtained which allows the exploitation of the special structure of the problem because realizations adapted to the structure of the arrays and functions can be selected in the next phase.

Finally, the **instantiation** phase [4] takes into account both the parameters of the IVP and of the target machine in order to: 1) schedule the tasks and allocate processors to each task, 2) select the best realization for each concept reference of the specification and the most suitable data distribution parameters for each realization chosen and 3) insert the required data redistribution routines. The goal is to obtain a good global runtime solution. This last phase can be performed systematically to obtain a description called *parallel frame program* which can be translated into a Fortran program augmented with MPI routines.

3 Functional Composition

To illustrate this phase, we employ an advanced numerical method to solve stiff ODEs [9], which is a parallelization across the Radau IIA method with 4 stages [3]. The analysis of this numerical scheme [4], makes it possible to derive an improvement of this scheme (called ONPILSRK) which presents a lower computational cost and exhibits greater task parallelism. We have selected several general concepts to specify this method. For instance, `LUdecomp`$(.., A, ..)$ denotes the LU Factorization of A, `SolveSystem`$(.., A, ..., X)$ denotes the computation of

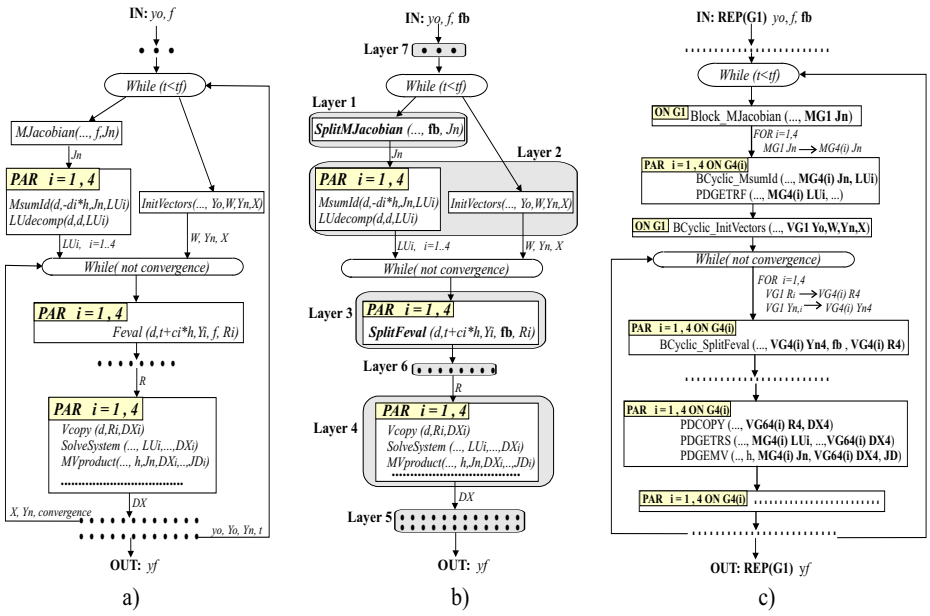


Fig. 3. a) General specification of the ONPILSRK method, b) Layered specialized specification ONPILSRK-VORTEX, c) Parallel frame program

$X \longleftarrow A^{-1}X$ (assuming `LUdecomp`$(.., A, ..)$) and the `Feval`$(.., t, f, y, dy)$ concept denotes the evaluation of the function f. These operations are the main sources of data parallelism exhibited by the method. To combine these concepts, we can use a graphical notation which expresses the concurrent and sequential composition as a task directed graph. A summarized description of the definitive general specification of the scheme is presented in Figure 3a, where the edges denote data dependencies between operations and in which the main sources of task parallelism in the method are represented with concurrent loops ($PAR\ i = 1, 4$).

4 Specialization

To illustrate this phase, we adapt the ONPILSRK method to two stiff IVPs:
• An IVP, noted as VORTEX, which models the evolution of two singular vortex patches governed by a two-dimensional incompressible fluid flow [1]. Assuming each vortex patch boundary has been parameterized by using $N/2$ points, we obtain a system with 2N ODEs where the evaluation of the equations of a point depends on all the other points (it involves a dense Jacobian) and the system function is cost-separable. The specialization of the functional specification for this problem consists of incorporating the cost-separable property of the system function f. This includes 1) defining a partitioning function **fb** as a sequential routine, and 2) substituting the general `Feval` and `MJacobian` concepts by specialized concepts which manage cost-separable functions as shown in Figure 3b. This enables the exploitation of the data parallelism existing in the evaluation of f by selecting specialized parallel implementations in the instantiation phase.

• An ODE system, noted as BRELAX, which describes a 1D rarefied gas shock [6]. The resulting system has 5N ODEs when the 1D space is discretized by using N grid points. The system function is also cost-separable and has a narrow banded structure with lower bandwith $Lw = 9$ and upper bandwidth $Up = 7$. To obtain the specialized specification for this IVP, several general concepts, which deal with matrices maintaining the banded structure of the Jacobian (for instance, `LUdecomp`) must be replaced by specializations which assume a banded structure for these matrices. The concepts `MJacobian` and `Feval` must be replaced by specializations which assume a cost-separable banded system function.

5 An approximation method to perform the instantiation

We propose an approximation method to perform the last phase, which assumes that these algorithms exhibit a task parallelism with a layered structure [8]. The method selects the most suitable realizations and data distributions in an ordered way, by giving priority to the most costly layers in the layered specification.

Following this method, we have obtained parallel implementations of the ONPILSRK scheme adapted to the VORTEX and BRELAX IVPs for the $P = 4$ and $P = 8$ nodes of a PC cluster with a switched fast Ethernet network. To illustrate the method, the instantiation of the ONPILSRK-VORTEX specification for $P = 8$ will be considered. Figure 3c presents a graphical description of the parallel frame program obtained. The nodes can represent a realization call or parallel loops applied on a sequence of calls. Every call is assigned to a processor group and some edges are labelled with redistribution operations [4].

The method uses the task layer definition which is introduced in [8] to identify layers in the graph. The layers are ordered taking into account the estimated execution time of the sequential realizations linked to the nodes of each layer, and then the layers are instantiated in an ordered way starting with the most costly ones. Figure 3b shows how the layers have been ordered in our example.

The instantiation of a layer depends on its internal structure and is based on the constrained instantiation of a concept reference. Given a layered functional graph, where several nodes may have been instantiated, and a reference to a concept C which belongs to a layer, the *constrained instantiation* of C on P_m processors, consists of selecting (R, P_R, D), where R represents a realization linked to C, $P_R \leq P_m$ denotes a number of processors and D is a set of parameters describing the distribution of the array arguments of R on P_R processors. This selection is performed such that the estimated execution time for the operation C is minimized taking into account the cost associated with the redistribution operations needed to adapt the data distributions of R (given by D) with the distributions followed in previously instantiated nodes. The constrained instantiation also involves inserting redistribution operations to adapt the data distributions.

When a layer encapsulates a chain of concept references, it is a **linear layer**. To instantiate a linear layer, its nodes are sorted according to the estimated sequential time and the constrained instantiation of each reference concept on the total number of processors is carried out starting with the most costly nodes

in the chain. In our example, the instantiation starts with the linear layer 1 (see Figure 3b) (the most costly one) which only contains one operation. Since there is no fixed data distribution which affects this layer, the optimal realization for the group with the 8 processors ($G1$) is chosen. This realization generates a block column distribution of the Jn matrix (distribution type `MG1`).

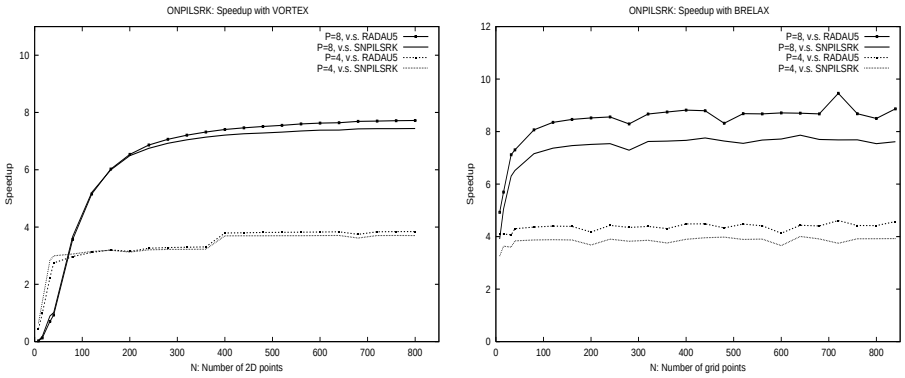
When a layer includes several concurrent chains, it is a **concurrent layer**. To instantiate a concurrent layer, all the possibilities of scheduling the chains are evaluated. To evaluate a scheduling choice where there are concurrent chains, the total number of processors is divided among the resultant concurrent chains, in proportion to the estimated sequential time of each chain. Next, every chain is instantiated as in a linear layer but on the previously assigned number of processors. The result of instantiating each chain is evaluated by combining the runtime formulas of the realizations selected and the redistribution costs. Finally, the choice which minimizes the estimated runtime for the layer is chosen.

In our example, layer 2 is a concurrent layer with 5 chains. The scheduling choice which gives the lowest cost consists of assigning each chain of the concurrent loop to disjointed groups with 2 processors ($G4(i)$, $i = 1, ..., 4$) and then executing the other one on the $G1$ group. This involves the use of a ScaLAPACK realization (LU factorization) [2], which assumes a 64×64 block-cyclic distribution of the matrix ($MG4(i)$, $i = 1, ..., 4$). Therefore the selection of this choice must take into consideration the redistribution of the matrix Jn. To instantiate layer 3, a realization to perform a parallel block evaluation of the function system which uses the partitioning function is chosen. The remaining layers are instantiated in an ordered way but taking into account the fixed distributions.

We have compared the runtime of the parallel programs obtained with two sequential solvers: *RADAU5* [3], an efficient stiff ODE solver, and the sequential implementation of the ONPILSRK scheme, called here SNPILSRK (see Figure 5). The implementations for the VORTEX IVP achieve a speedup of 3 to 3.7 on 4 processors and a speedup of 6.5 to 7.5 on 8 processors with regard to SNPILSRK. The implementations for BRELAX achieve a speedup of 3.65 to 4 on 4 processors and a speedup of 7.15 to 7.85 on 8 processors with regard to SNPILSRK. The results obtained with regard to RADAU5 are slightly better.

6 Conclusions

The COMPODES approach to deriving parallel ODE solvers is proposed. The approach makes it possible to manage several implementations of the operations and enables the separate treatment of different aspects during the derivation:
a) *Functional aspects*: A generic description of the functionality and the task parallelism of the method is obtained in the initial phase by focusing on optimizations independent of the machine and the problem.
b) *Problem structure*: In the *specialization* phase, all the structural characteristics of the problem are integrated into the method description in order to enable its exploitation in the next phase.
c) *Performance aspects dependent on both the architecture and the problem*: A simple approximation method has been proposed to take systematically all the



a) b)

Fig. 4. Speedup Results for a) VORTEX IVP and b) BRELAX IVP

parallel design decisions which affect the performance of the final program by considering both the parameters of the problem and the machine.

The approach is illustrated by deriving several efficient implementations of a stiff ODE solver for a PC cluster. Following this approach, satisfactory results have been obtained in the solution of two different problems.

Acknowledgements

This work was supported by the projects TIC2000-1348 and BFM2002-01710 of the Ministerio de Ciencia y Tecnología.

References

1. Carrillo, J. A., and Soler, J.: On the Evolution of an angle in a Vortex Patch. The Journal of Nonlinear Science, 10:23–47, 2000.
2. Dongarra, J., Walker D.: Software libraries for linear Algebra Computations on High Performance Computers. SIAM Review, 37(2):151–180, Jun. 1995.
3. Hairer, E., Wanner., G.: Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems. Springer-Verlag, 1996.
4. Mantas, J. M., Ortega, J., Carrillo J. A.: Component-Based Derivation of a Stiff ODE Solver implemented on a PC Cluster. International Journal of Parallel Programming, 30(2), Apr. (2002).
5. Mantas, J. M., Ortega, J., Carrillo J. A.: Exploiting the Multilevel Parallelism and the Problem Structure in the Numerical Solution of Stiff ODEs. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, (2002).
6. Mantas, J. M., Ortega, J., Pareschi, L., Carrillo J. A.: Parallel Integration of Hydrodynamical Approximations of the Boltzmann Equation for rarefied gases on a Cluster of Computers. Journal of Computational Methods in Science and Engineering, JCMSE, 3(3):337–346, 2003.
7. Ramaswamy, S., Sapatnekar, S., Banerjee, P.: A Framework for Exploiting Data and Functional Parallelism on Distributed Memory Multicomputers. IEEE Trans. Parallel and Distributed Systems, 8:1098–1116, Nov. 1997.
8. Rauber, T., Rünger, G.: Compiler support for task scheduling in hierarchical execution models. Journal of Systems Architecture, 45:483–503, 1998.
9. Van der Houwen, P. J., de Swart, J. J. B.: Parallel linear system solvers for Runge-Kutta methods. Advances in Computational Mathematics, 7:157–181, Jan. 1997.