# Parallelization of Finite Element Codes with Automatic Placement of Communications

Laurent Hascoët

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Parallelization of finite element codes with automatic placement of communications*

Laurent Hascoët

**N° 3646**

Mars 1999

THÈME 4


ISSN 0249-6399

# Parallelization of finite element codes with automatic placement of communications

Laurent Hascoët

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Sinus

Rapport de recherche n° 3646 — Mars 1999 — 25 pages

**Abstract:** We present a tool dedicated to automatic SPMD parallelization of iterative mesh-based computations, and its application to existing codes. The tool automatically places communication statements into the program, to manage the necessary updates between overlapping parts of the partitioned mesh. It is parameterizable with a description of the type of overlapping between sub-meshes. We present an application of this tool to two-dimensional and three-dimensional Navier-Stokes flow solvers. Performance results are given.

**Key-words:** parallelization, SPMD, program analysis, program transformation, mesh partition, finite elements, Navier-Stokes

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

# Parallélisation de codes éléments finis avec un placement automatique des communications

**Résumé :** Nous présentons un outil destiné à la parallélisation SPMD automatique de programmes de résolution itératifs basés sur des maillages. Nous présentons la conception de l'outil et son utilisation. Cet outil insère dans le programme les communications nécessaires pour tenir à jour les valeurs aux frontières des sous-maillages. Il est paramétrable grâce à une description du type de recouvrement choisi entre les sous-maillages. Nous présentons l'application de cet outil à des solveurs Navier-Stokes à deux et trois dimensions, et nous donnons des mesures de performances.

**Mots-clés :** parallélisation, SPMD, analyse de programme, transformation de programme, partition de maillage, éléments finis, Navier-Stokes

# Contents



# 1 Introduction

For many application programs, parallelization is an appealing, but delicate, way to improve performance. This is especially true in scientific computing. In this domain, one category of programs deserves special attention. These are the iterative computations on unstructured meshes. They are very widespread, and consume a significant part of the computational resources. These programs are natural candidates for parallelization.

There exist many parallelization techniques. This is due, to a large extent, to the numerous machine architectures. As the art evolves, a matching emerges between the classes of programs and the parallelization techniques.

For example, computations on unstructured meshes are now generally parallelized through a geometric partitioning of the mesh itself. The parallelized program uses the SPMD (*Single Program Multiple Data*) model. In this model, each processor of a distributed memory computer runs the same program on one particular portion of the original mesh. Communications are required when a value must travel from a sub-mesh to another. Usually, this happens only for values located on the border of sub-meshes. It is common practice to duplicate a certain amount of these "border" values on the neighboring sub-meshes. This allows us to defer communications until all the duplicated values are out of date and need to be updated again.

The adaption of a program to this SPMD parallelization strategy requires two main operations: *Mesh Partitioning* and *SPMD Program Generation*. These will be discussed in detail in section 3. Writing the SPMD program by hand is a delicate task. It requires knowledge of the application to find the places where values travel between sub-meshes. It also requires knowledge of the overlapping between sub-meshes, to correctly update duplicated values. Therefore, we propose a tool that generates the SPMD program mechanically. It is important to note that this is a specialized parallelization tool: this tool makes sense only for computations on unstructured meshes. This is our deliberate choice, to trade generality for efficiency on a precise domain. Our justifications are that:

- computations on unstructured meshes are a very frequent kind of programs. Generic parallelizers behave poorly due to the intensive use of indirection arrays.
- the knowledge of how meshes are structured, and how programs behave on meshes, allows an automatic choice of optimal communications insertion. This would be out of reach of a generic parallelizer, because it cannot infer this knowledge from the program.

In this paper, we present a tool for automatic generation of a SPMD program for unstructured meshes. This tool determines the optimal locations of communication calls, to give best performance of the parallelized program. This tool is based on the method and algorithms presented in [4], that we shall summarize in section 4. We also show here, how this tool was used to parallelize

existing 2D and 3D Navier-Stokes solvers. The rest of the paper is organized as follows: Section 2 briefly introduces the original sequential Navier-Stokes solvers. Section 3 presents precisely the SPMD model of parallel execution. Section 4 describes the principle of our automatic SPMD program generation. Section 5 explains the specific problems related to the "real" programs, and how they were solved. Section 6 presents the application of our tool to the Navier-Stokes solver, and the generated SPMD program. Section 7 compares the present method with related techniques.

# 2 The *Exp2D*, *Imp2D*, and *NSC3DM* Navier-Stokes solvers

Our application examples are two-dimensional and three-dimensional flow solvers, that we shall refer to as *Exp2D*, *Imp2D*, and *NSC3DM*. They solve the compressible Navier-Stokes equations using a mixed finite element/finite volume method, designed on unstructured triangular meshes. Steady-state solutions are obtained using a pseudo-transient approach which is based on a linearized formulation. For *Exp2D*, this formulation is *explicit*, whereas for *Imp2D* and *NSC3DM*, the formulation is *implicit*. Therefore in *Imp2D* and *NSC3DM*, each time step requires the solution of a large sparse linear system. Approximate solutions of these systems are obtained by using Jacobi relaxations.

From a programming point of view, it turns out that the most frequent operations on the mesh are of the type known as *Gather-Scatter*. These operations consist of a loop on all the mesh elements of a given kind, say, triangles. For each triangle, the mesh structure gives a direct access to its three nodes. The operations on this triangle are decomposed in three phases:

1. **Gather:** some values attached to the three nodes are read.
2. **Compute:** these values are used, with others, to compute three partial results. Those are the contribution of the triangle to the total results of its nodes.
3. **Scatter:** each partial result is accumulated into the total result of the corresponding node.

At the end of the loop on triangles, each node will hold the correct final result, i.e. the accumulation of the partial results of all its neighboring triangles.

Table 1 gives a rough idea of the size of these codes. Comments are not counted in the number of lines.

| Solver | *Exp2D* | *Imp2D* | *NSC3DM* |
|---|---|---|---|
| subroutines | 13 | 18 | 34 |
| lines | 1500 | 1800 | 4800 |

Table 1: Sizes of application examples

# 3 The SPMD parallel execution model

As we saw in section 1, the SPMD parallel execution model requires two major operations: *Mesh Partitioning* and *SPMD Program Generation*. As shown on figure 1, these operations are independent. However, both require information about the pattern of duplicated elements (*overlapping pattern*). Also, we shall see that the particular SPMD program style used here, requires that the partitioner represents sub-meshes in a precise manner.

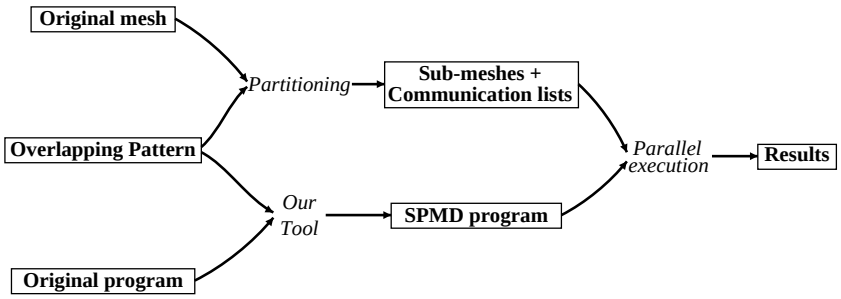


Figure 1: *SPMD parallelization process*

## 3.1 Mesh Partitioning

The initial mesh must be partitioned into as many sub-meshes as there are available processors. Any mesh partitioner can do that, provided it respects the constraints given below.

A variety of techniques, such as the recursive bisection method, are used to minimize the size of the interface between sub-meshes, therefore minimizing the need for expensive communications. But, whatever mesh partition we end up with, this will not affect the result of our SPMD program generator. Only, the resulting SPMD program will perform better on a better mesh partition. The partitioner is entirely in charge of dealing with *load balancing*. When one processor is over-used or under-used, or when *mesh refinement* changes the number of mesh elements in sub-meshes, the partition must be updated. This task belongs to the mesh partitioner. But the SPMD program itself need not be modified.

Following the decision of the user, the partitioner must create a certain number of overlapping sub-mesh elements. This will allow many communications to be gathered, yielding a dramatic reduction of communication overhead. This requires the partitioner to pre-compute and store the lists of overlapping mesh elements, between any two given sub-meshes. These lists are later used by the communication routines. It is worth comparing this approach with the classical "*inspector-executor*" paradigm. Anticipating on section 7, we remark that the present method amounts to leaving the "*inspector*" task to the mesh partitioner.

For our application, all there is to choose is the amount of overlapping between sub-meshes, that we call the *overlapping pattern*. In two dimensions, we chose to set a one-triangle wide overlapping zone, as shown on figure 2. Similarly in three dimensions, we set a onetetrahedron wide overlapping zone. This choice will strongly affect the placement of communications by our tool. Therefore, because of overlap and communication cost, this will affect the performances of the resulting SPMD program. In [3], one can find a discussion comparing various overlapping patterns. The best candidates seem to be:

1. One layer of overlapping triangles, with neighboring edges and nodes (*cf* figure 2) .
2. Only one layer of overlapping nodes (*cf* figure 4).