Comparison of SUPG and FVG techniques on KSR-1

Bijan Mohammadi

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

Comparison of SUPG and FVG techniques on KSR-1

Bijan Mohammadi

N° 2357

Octobre 1994

PROGRAMME 6

Calcul scientifique,

modélisation

et logiciel numérique

ISSN 0249-6399

Comparison of SUPG and FVG techniques on KSR-1

Bijan Mohammadi

Programme 6 — Calcul scientifique, modélisation et logiciel numérique
Projet MENUSIN

Rapport de recherche n° 2357 — Octobre 1994 — 17 pages

Abstract: Different parallel strategies for solving compressible Navier-Stokes equations using Finite-Volume-Galerkin (FVG) and Streamline Upwind Petrov-Galerkin (SUPG) techniques on unstructured grids are presented. Particular attention is brought to the parallelization on KSR-1. Tiling technique is compared to parallel regions. Results are also compared to those obtained on serial workstations.

Key-words: Parallel Computing, CFD, SUPG, FVG, Tiling, Parallel Regions.

(Résumé : tsvp)

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Comparaison des techniques SUPG et FVG sur KSR-1

Résumé : On présente differentes stratégies parallèles pour résoudre les équations de Navier-Stokes compressibles utilisant un schéma décentré Petrov-Galerkin (SUPG) ainsi qu'une technique Volumes-Finis-Galerkin (FVG) sur des maillages non-structurés. Une attention particulière est apportée à la parallélisation sur la KSR-1. Les approches *"tiling"* et par sous-domaines sont comparées. Les résultats sont aussi comparés à ceux obtenus sur des stations de travail scalaires.

Mots-clé : Calcul Parallèle, CFD, SUPG, FVG, Tiling, Régions Parallèles.

1 Introduction

The aim of this work is to present and compare different approaches for parallelizing fluid dynamics codes working on unstructured meshes. The numerical methods used here are based on either an Upwinded Finite-Element method, like SUPG [HUG 1,2,3] or Roe-Deconinck [RD1] technique, or a combination of Finite-Volume and Finite-Element methods called Finite-Volume-Galerkin [DE1].

The difficulty in the parallelization of these techniques comes from the fact that different coupled structures are involved. This coupling introduces communications and collisions which are difficult to avoid if automatic parallelization is used. In fact we are interested in using as much as possible the automatic parallelization tools available in KSR-1. This should be portable on other architectures which propose a shared memory distribution.

One possibility to reduce communications is to simplify the data structure involved in the computation by reducing the number of different entities which interact in the structure. For instance the Finite-Element approach needs two geometrical entities called nodes and elements while the Finite-Volume-Galerkin technique requires also informations on the edges. The first class of methods therefore requires less informations and is an element-by-element technique. We can notice that this is why Finite-Difference techniques are easier to parallelize. In fact in these techniques only nodes are involved and the weighting procedure for meshes is a priori known and is invariant.

The first part of the paper is devoted to the evaluation of the automatic parallelization technique available in KSR-1 called *"tiling"* technique for our problem. The domain decomposition approach which is called *parallel regions* on KSR-1 has also been tested. This technique is rather classical and is portable on all MIMD architectures and even on clusters through a message passing library like PVM.

In the second part, we have compared our tile based and parallel regions based solver to have a more precise idea on the differences between tiling and parallel regions techniques and also to see the limits of tilings for these applications. We have also compared these solvers to the parallel regions based Finite-Volume-Galerkin code of Stephane Lanteri [LAN1].

In section 2 we give a summary description of the governing equations. Section 3 is devoted to the description of the different parallel models and corresponding results. Through these examples we have tried to cover a large range of grain sizes going from small (20000 variables) to large (4 millions variables).

2 Governing Equations and Numerical Techniques

Let ρ be the density, $u = (u_1, u_2)$ the velocity, T the temperature, $E = T + \frac{\|u\|^2}{2}$ the total energy, $p = (\gamma - 1)\rho T$ the pressure, $\nabla u = u_{i,j}$ the gradient of u, $D = u_{i,i}$ its divergence, $S = (\nabla u + \nabla u^t) - \frac{2}{3} DI$ the deformation rate tensor. The Navier-Stokes equations in nondimensional form are:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) &= 0 \\ \frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho u \otimes u) + \nabla p &= \nabla \cdot (\mu S) \\ \frac{\partial \rho E}{\partial t} + \nabla \cdot ((\rho E + p) u) &= \nabla \cdot (\mu S u) + \nabla (\kappa \nabla T) \end{aligned} \tag{2.1}$$

with

$$\kappa = \frac{\gamma \mu}{Pr}, \quad \gamma = 1.4, \quad Pr = 0.72.$$

Here $\mu = 1/Re$ is the inverse of the Reynolds number.

These equations describe the newtonian fluids behaviour. We need to associate with (2.1) a set of initial and boundary conditions. The initial condition can be a uniform state. Infinite boundaries are treated by a characteristic technique. This means that along these boundaries the fluxes are splitted in positive and negative parts following the sign of the eigenvalues of the jacobian of the convective part of (2.1) and the negative incoming part is imposed. Moreover, on these boundary the viscous effects are supposed to be negligible. On the solid boundary, we assume no-slip condition for u and specified temperature. Sliping boundary condition is necessary for Euler computations or sometimes in viscous cases. This condition is applied on solid boundaries for

Euler computations and on symmetry lines depending on the physic of the problem.

The convective part of the Navier-Stokes equations are solved by either a Finite-Volume-Galerkin [DE1][FE1,2] technique or by a Streamline Upwind Petrov-Galerkin scheme [HUG1,2,3] and the viscous terms are treated by using a standard Galerkin technique. Our algorithm is explicit in time and uses an iterative scheme. More details about the code can be found in [MO1].

3 Parallel Models

This section is devoted to the description of the different parallel strategies which were tested in this work for solving the compressible Navier-Stokes equations.

In what follows, $is = \{1,..,ns\}$ will denote a node, $it = \{1,..,nt\}$ an element and $iseg = \{1,..,nseg\}$ an edge of a mesh.

3.1 Tiling

One of our goals in this work is to use the tools available in KSR-1 for parallelizing codes. The simplest parallel technique is called *tiling*. Tiling can be seen as an automatical parallelization of nested loops by adding directives to existing programs. Tiling enables the user not to take care of subdomains definition which is one of the hardest task in parallel computation on unstructured grids. But this technique is really efficient only when one independant data structure is involved as in the following example, which is often given as tutorial:

```
integer is,ns

c*ksr*tile(is,numthreads=(num_proc),private(...))
do is = 1 , ns
 do_work_on_(is)_using_information_from_(is)
 ...
enddo
c*ksr*endtile
```

Here no communication is necessary and the part of the job affected to each processor is easy to define. The difficulty of CFD in Finite-Element context is that several (at least two) different structures are involved. These entities are called nodes, elements and edges. By this we mean that informations might be available on nodes, elements or edges. The following example show such a situation where node and element informations interact:

```
integer is,ns,it,nt

c*ksr*tile{it,numthreads=(num_proc),private(...)}
do it = 1 , nt
 do_work_on_elements(it)_using_information_from_nodes
 ...
enddo
c*ksr*endtile

c*ksr*tile{is,numthreads=(num_proc),private(...)}
do is = 1 , ns
 do_work_on_nodes(is)_using_information_from_elements
 ...
enddo
c*ksr*endtile
```

This example shows that when different coupled data structures are involved, communications and collisions might make the use of tiling more questionable. We also expect that as it is not possible to make all the code parallel by the tiling technique, the ratio of serial to parallel degrades the overall performances of the code.

Remark :

The situation where only two entities (here nodes and elements) are involved is the best but usually schemes for unstructured meshes require some other informations. For instance, the Finite-Volume-Galerkin method, classically used at INRIA [FE1,2][DE1] requires also informations on the edges. In

fact, for first order spatial accurate Euler computations with FVG, only two structures are needed (i.e. nodes and segments). The difficulty comes with the computation of the gradients. Therefore if an a priori weighting procedure is known for a given mesh (as for instance in a five or nine points Finite Difference technique), a segment-by-segment assembling can be used for the gradient computation.

As irregular meshes are characterized by their number of elements and as the Finite-Element method is clearly an element-by-element technique, the choice of the element as parallel data structure seems to be natural. □

Above we have presented the difficulties coming with different coupled data structures. An example of situation where this happens is given by our iterative algorithm, presented below, where at each iteration, we first assemble the right hand side of our streamline upwind system doing a loop on elements and scattering the contribution of each elements to nodes and then update the solution. This is done in serial by (nen being the number of elemental nodes and nu(nen,nt) the connectivity table):

```
do is=1,ns
sol(is)=sol0        ! solution initialization
rhs(is)=0.0
enddo

do iteration = 1 , number_of_iterations

do it = 1 , nt
 do ien=1,nen
 is=nu(ien,it)
 ...
 compute contribution_(ien,it)
 ...
 rhs_(is) = rhs_(is) + contribution_(ien,it)    ! scatter
```

```
 enddo
enddo

norm = 0.0
do is = 1 , ns
 sol(is) = sol(is) + dtl(is) * rhs(is)    ! solve du/dt=rhs
 norm=norm+abs(rhs(is))                   ! residual computation
 rhs(is) = 0.0                            ! rhs initialization
enddo

if(norm < epsilon) stop         ! solution is reached

enddo      !  end iterations
```

There are four parts in this algorithm: initialization, right-hand side assembling, scatter and solution updating by an explicit resolution. From a scalar point of view, the work to do is clearly concentrated in the assembling part. As we saw, there is coupling between element and node informations. To reduce communications, we need a good locality of the data. A renumbering procedure has therefore been used to improve this locality. It is based on the following heuristic : start from an initial element then define wavefronts by renumbering neighboring elements and nodes. This algorithm looks like a Cuthill McKee or a frontal algorithm.

Using *tile* parallel directives the assembling and resolution parts of the previous code become:

```
c*ksr*tile(it,numthreads=(num_proc),private=(ien,is,...))
do it = 1 , nt
 do ien=1,nen
 is=nu(ien,it)
 ...
 compute contribution_(ien,it)       ! rhs element-by-element
 ...
 rhs(is)=rhs(is)+contribution_(ien,it) !  scatter
```

```
 enddo
enddo
c*ksr*end tile

c*ksr*tile(is,numthreads=(num_proc))
do is = 1 , ns
 sol(is) = sol(is) + dtl(is) * rhs(is)       ! solve du/dt=rhs
enddo
c*ksr*end tile
```

We have tested this approach for inviscid and viscous supersonic flow over a cylinder at Mach 3. Iso-Mach contours for the inviscid case are shown in Fig. 1.

Time measurements have been done for 100 iterations in time of the viscous computation using the SUPG scheme in space presented above. As only parallel directives have been added to the code, the same code has been tested on different workstations (in serial) and results are also reproduced here. Test case configurations are shown below, the number of variables treated going from 20000 to 2 millions.

TEST CASES CONFIGURATIONS

CASE	1	2	3	4	5	6
ns	5000	10 000	50 000	100 000	500 000	1 000 000
nt	9702	19602	98 802	198602	997 002	1 996 002
nvar	20000	40000	200 000	400 000	2 000 000	4 000 000

The following two tables give the time spent in the assembling and explicit resolution parts.

Several remarks can be made:

First, the processor of the KSR-1 is somehow slow compared to the workstations.

Second, the speed up for the assembling part degrades faster when the grain is small, which is logical.

Third, the coupling mentioned before degrades the global speed up. This stays stable when the number of processors grows.

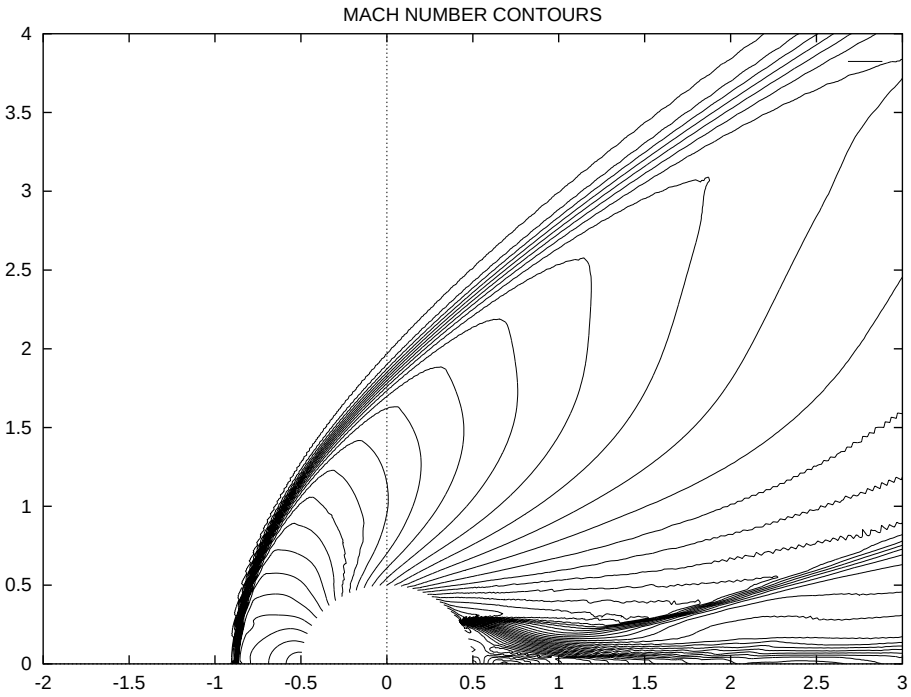

Figure 1: Inviscide Flow over cylinder at Mach 3, Streamline Upwind computation.

The first two remarks leads to a paradox in the sense that if we want performance on KSR-1 for the assembling part, we need quite a large grain and this is incompatible with its processor speed. The third point proves that the tiling technique for this problem reachs fast its limit.

TIME FOR RHS ASSEMBLING AND SCATTER VS. NB. PROC.

PROC.	1	2	4	8	16	32	64	HP735	SUN10
CASE 1	33	23	13	7	7.1	7.5	9	10	36
CASE 2	67	44	24	12	11	11	10	23	73
CASE 3	343	178	127	68	49	58	51	122	391
CASE 4	NC	479	262	137	104	119	105	250	761
CASE 5	NC	NC	1292	695	838	633	583	1264	
CASE 6	NC	NC	NC	2603	1450	1204	1100	2505	

Remark: NC configurations have also been computed but due to the amount of memory available the results are particularly slow.

The tiling technique works well from the load-balancing point of view and KSR-1 seems to correctly assign the same amount of work to all the nodes but concerning the scalability, the time evolution with the size of the problem and the number of processors is not always clear. These results are slightly better than those obtained in [FP1] for a laplacian assembling and an explicit resolution in the context of wave equation. This seems to be due to the fact that the amount of work to do is much higher in this case and this leads to a larger grain.

TIME FOR EXPLICIT RESOLUTION VS. NB. PROC.

PROC.	1	2	4	8	16	32	64	HP735	SUN10
CASE 1	3.7	2.6	1.8	1.2	1.1	0.8	1.3	2	12
CASE 2	7.5	4.5	3.7	2.3	2.1	1.4	1	5	22
CASE 3	38	25	18	10	5	5	6	26	111
CASE 4	NC	73	37	20	16	13	16	53	211
CASE 5	NC	NC	188	105	88	56	36	268	
CASE 6	NC	NC	NC	1025	241	108	66	601	

In the previous assembling procedure, no care has been taken for collisions when computing the rhs. This of course makes operations slower. To avoid that we introduce a local array in the assembling part and make the scatter separately as follows:

```
c*ksr*tile(it,numthreads=(num_proc),private=(ien,is,...))
do it = 1 , nt
 do ien=1,nen
 is=nu(ien,it)
 ...
 compute contribution_(ien,it)     ! rhs element-by-element
 ...
 enddo
enddo
c*ksr*end tile

c*ksr*tile(it,numthreads=(num_proc),private=(ien,is))
do it=1,nt
 do ien=1,nen
  is=nu(ien,it)
   rhs(is) = rhs(is) + contribution_(ien,it)   ! scatter
 enddo
enddo
c*ksr*end tile
```

The assembling times obtained by this approach are somehow similar to the previous ones. In fact, the assembling times have almost not changed while the scatter times are growing so that the global times (assembling+scatter) are greater than before. To be really efficient we need to avoid all the collisions and for this we have to make available all the informations located at nodes to elements. This is only possible by duplicating any array $A(ns)$ by introducing another array $AT(nen, nt)$ by:

```
    do it=1,nt
```

```
      do ien=1,nen
        is=nu(ien,it)
        AT(ien,it)=A(is)
      enddo
    enddo
```

which is not reasonable as we saw that scatter takes too much time. The classical technique to avoid collision is the coloring technique but this can not work either in our case as we need quite a large grain and as few communications as possible.

Remark :

Another problem, which is perhaps a characteristic of parallel computers, comes with I/O operations. This makes the use of these machines for practical applications difficult as I/O plays an important role. Another disappointing point being that even in a parallel regions approach the I/O operations remain sequential on the KSR-1.

3.2 Parallel Regions

The parallel regions technique is a classical MIMD programming technique. Therefore it is also the most convenient to export to other MIMD architectures and to cluster of workstations using a message passing library like PVM. The difficulty when using this technique in our case is that to be efficient, we need a decomposition of the physical domain in subdomains having a good locality. The interface between two subdomains should be as small as possible to minimize communications. Once the subdomains are defined, we identify the processors we need and do the job locally.

```
call ipr_create_team (num_proc, idteam)

c*ksr*parallel region (teamid = idteam, private = (...))

  do it=1,nt_loc
    ...
```

```
    do local assembling on each processor
      ...
   enddo

c*ksr*end parallel region

   communicate interface contributions between processors

c*ksr*parallel region (teamid = idteam, private = (...))

   do is=1,ns_loc
     ...
    do local resolution
     ...
   enddo

c*ksr*end parallel region
```

To minimize inter-processors communications, the interfaces between subdomains have to be minimized and this is the purpose of automatic partitionning tools.

In this part, we compare the tiling and parallel regions techniques available in KSR-1 for our SUPG scheme. We also compare these results to those obtained with the parallel regions based Finite-Volume-Galerkin solver of Stephane Lanteri [LAN1]. The test case consists of a transonic flow over a NACA 0012 at Mach number 0.85 and Reynolds number 1000. Two levels of discretization have been used (see table below).

CASE	6	7
ns	8119	32226
nt	15998	63992
nvar	32476	124504

Fig. 2. shows the mesh and its partitionning in subdomains. The following two tables show the speed up obtained using different parallel strategies for the SUPG scheme and the Finite-Volume-Galerkin scheme.

Case 6 is a small case. We can see that as previously both assembling and resolution speed up for the tiling technique degrade fast. The total time (assembling + resolution) clearly shows the stagnation of the tiling technique when the number of processors grows. Also, as for this case the grain is quite small the speed up for all the methods is limited.

TOTAL AND ASSEMBLING SPEED UP VS. NB. PROC. FOR CASE 6

NB. PROC.	1	4	16
TOTAL FVG PAR. REG.	1	3.1	8.25
TOTAL SUPG PAR. REG.	1	3.2	8.3
TOTAL SUPG TILE	1	2.6	4.2
ASSEM FVG PAR. REG.	1	3.0	8.9
ASSEM SUPG PAR. REG.	1	3.1	9.3
ASSEM SUPG TILE	1	2.7	6.15

For case 7 the situation is quite similar. The parallel regions speed up are better because the size of the grain is bigger.

TOTAL AND ASSEMBLING SPEED UP VS. NB. PROC. FOR CASE 7

NB. PROC.	1	4	16
TOTAL FVG PAR. REG.	1	4.1	14
TOTAL SUPG PAR. REG.	1	4.0	13.5
TOTAL SUPG TILE	1	2.6	6.8
ASSEM FVG PAR. REG.	1	4.1	15.4
ASSEM SUPG PAR. REG.	1	3.96	14.8
ASSEM SUPG TILE	1	2.69	6.8

Again all processors have an equal amount of work when using the tiling technique while for parallel regions this is true only if the user-defined subdomains are suitable.

These cases show that the parallel regions technique is more suitable for applications where different data structures interact and this even if we have reduced the number of communications using a SUPG scheme. For the tiling strategy to be performant in these cases, the shared memory access have to

be much faster or the data locality has to be quite good. Other renumbering procedure improving the locality might help. But this is like doing parallel regions. In all cases, this will not be anymore automatic parallelization.

4 Summary and Concluding Remarks

The implementation of an SUPG scheme on the KSR-1 for inviscid and viscous compressible flows has been presented. Both tiling and parallel regions strategy have been tested. These approachs have also been compared to a parallel Finite-Volume-Galerkin code using parallel regions. The results have also been compared to those obtained on different workstations.

Our major conclusions are :

* The parallelisation using the tiling technique proposed on KSR-1 is easy, the drawback being the non-portability of the code on architectures without a shared memory distribution.

* Load-balancing is well managed by the tiling technique while the scalability of the applications is less clear, specially for these cases where different structures interact. For parallel regions on the other hand, the scalability behaviour is better. It is important to notice that for these applications, scalability is more important than load balancing. A reasonable load-balancing is sufficient.

* To get really good performance the grains have to be quite big but the KSR-1 cpus are not so fast.

* Tiling strategy seems to be not suitable for Finite-Element applications because the shared memory access is not fast enough. Also the locality of the data has to be perfect. For this you need a subdomain definition. But then, this means doing parallel regions. A parallel regions approach therefore seems to be more convenient.

Acknowledgements

The Finite-Volume-Galerkin results presented here have been obtained with the parallel regions based code of Stephane Lanteri of INRIA-SINUS at Sophia Antipolis-France.

The renumbering routines have been kindly made available to us by Jocelyne Erhel of IRISA-ALADIN at Rennes-France.

5 References

[KSR] KSR User Guides for Parallel Programming.

[DE1] A. Dervieux,*"Steady Euler Simulations using Unstructured Meshes"*, Von Karman Lecture notes series, Mars 1985.

[FE1] L. Fezoui, A. Dervieux, *"Finite Element Non Oscillatory Schemes for Compressible Flows"*, Comp. Math. and Applic. 8th France-U.S.S.R.-Italy Joint Sympos. Pavia. 1989.

[FE2] L. Fezoui, B. Stoufflet, *"A Class of Implicit Upwind Schemes for Euler Simulations with Unstructured Meshes"*, Journ. of. Comp. Phys., 84, pp. 174-206, 1989.

[FP1] P. Feat,*"Parallélisation d'un algorithme de résolution de l'équation des ondes"*, Rapport de stage, IRISA 1993.

[HUG1] HUGHES T. J. R., FRANCA L.P., MALLET M.,*" A New Finite Element Formulation for Computational Flui d Dynamics: I. Symmetric Forms of the Compressible Euler and Navier-Stokes Equations and the Second Law of Thermodynamics"*, Computer Methods in Applied Mechanics and Engineering, num. 54, pp. 223-234, 1986.

[HUG2] HUGHES T. J. R., MALLET M.,*" A New Finite Element Formulation for Computational Fluid Dynamics: III. The Generalized Streamline Operator for Multidimensional Advective-Diffusive Systems"*, Computer Methods in Applied Mechanics and Engineering, num. 58, pp. 305-328, 1986.

[HUG3] HUGHES T. J. R., MALLET M.,*" A New Finite Element Formulation for Computational Fluid Dynamics: IV. Discontinuity Capturing Operator for Multi-Dimensional Advective-Diffusive Systems"*, Computer Methods in Applied Mechanics and Engineering, num. 58, pp. 329-339, 1986.

[LAN1] S. Lanteri, C. Farhat, *"Viscous Flow Computation on MPP Systems: Implementation Issues and Performance Results for Unstructured Grids "*, Proc. Sixt SIAM conf. on parallel Precessing for Scientific Computing, Norfolk, Virginia, pp. 65-70, 1993.

[MO1] B. Mohammadi, *"NSC2KE, An User Guide"*, INRIA technical report 164, May 1994.

[RD1] H. Pallière, H. Deconinck, R. Struijs, P.L Roe,, L.M.Mesaros, J.D. Muller,*"Computations of Inviscid Compressible Flows using Fluctuation-Splitting on trianglar"*, AIAA 93-3301.

Unité de recherche Inria Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 Villers Lès Nancy
Unité de recherche Inria Rennes, Irisa, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Unité de recherche Inria Rhône-Alpes, 46 avenue Félix Viallet, 38031 Grenoble Cedex 1
Unité de recherche Inria Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex
Unité de recherche Inria Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex

Éditeur
Inria, Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
ISSN 0249- 6399