Couplage de codes numériques, parallélisme et langages de haut niveau

François Clément, Arnaud Vodicka, Roberto Di cosmo, Pierre Weis

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Couplage de codes numériques, parallélisme et langages de haut niveau

François Clément — Arnaud Vodicka — Roberto Di Cosmo — Pierre Weis

N° 4825

Mai 2003

THÈMES 4 et 2

ISSN 0249-6399 ISRN INRIA/RR--4825--FR

Couplage de codes numériques, parallélisme et langages de haut niveau

François Clément [*] , Arnaud Vodicka [†] , Roberto Di Cosmo [‡] , Pierre Weis [§]

Thèmes 4 et 2 — Simulation et optimisation
de systèmes complexes — Génie logiciel
et calcul symbolique
Projets Estime et Cristal

Rapport de recherche n° 4825 — Mai 2003 — 33 pages

Résumé : Le couplage de codes numériques associés à différents sous-domaines permet la mise en œuvre de la modélisation de phénomènes physiques complexes. Les techniques de couplage s'apparentent à celles de parallélisation : les communications correspondent à des conditions de transmission aux interfaces entre les sous-domaines. Dans cette première évaluation du potentiel des langages de haut niveau pour le calcul scientifique, nous abordons la réalisation d'une maquette de couplage pour un cas simplifié en utilisant le langage fonctionnel Caml, et plus précisément à l'aide du système OCamlP3l dédié à la parallélisation.

Mots-clés : couplage multiphysique, décomposition de domaine, parallélisme, langage de haut niveau, Caml

Rapport d'activité durant la première année d'Ingénieur Associé d'A. Vodicka.

[*] Projet Estime. Francois.Clement@inria.fr.
[†] Projet Estime. Arnaud.Vodicka@inria.fr.
[‡] Projet Cristal et PPS, Université de Paris 7. roberto@dicosmo.org.
[§] Projet Cristal. Pierre.Weis@inria.fr.

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

Coupling of numerical codes, parallelism and high-level languages

Abstract: Coupling numerical codes associated to different subdomains allow the implementation of complex phenomena. Coupling techniques are similar to parallelization techniques: communications correspond to transmission conditions at the interfaces between subdomains. In this first study of the potential of high-level languages for scientific applications, we address the design of a coupling sample for a simplified case using the fonctional language Caml, and more precisely the OCamlP3l system devoted to parallelism.

Key-words: multiphysics coupling, domain decomposition, parallelism, high-level language, Caml

Table des matières

1 Introduction

La compréhension des écoulements et du transport en milieu poreux est à la base de la modélisation des problèmes d'environnement dans le sous-sol—gestion des ressources en eau, pollution, stockage de déchets—et de la simulation de réservoirs pétroliers. La simulation des écoulements et du transport en milieu poreux, à grande échelle et en trois dimensions, conduit souvent, et de plus en plus, à des problèmes de grande dimension qui ne peuvent être traités que par couplage de modèles appropriés associés à différents sous-domaines. C'est pourquoi le projet Estime étudie l'extension et l'implémentation de méthodes de décomposition de domaine sans recouvrement pour des couplages multiphysiques ou multinumériques.

La technique utilisée consiste à calculer l'écoulement et le transport en trois dimensions, dans un milieu à plusieurs couches, avec une source placée dans une boîte maillée avec un maillage plus fin qui ne se raccorde pas au maillage global. Le couplage consiste à faire communiquer durant un procédé itératif les différents sous-domaines (i.e., à faire communiquer la valeur d'un champ calculé par un sous-domaine à ses sous-domaines voisins) afin d'obtenir une solution globale sur tout le maillage. Les solveurs de sous-domaine sont écrits en Fortran, C ou C++ et sont déjà pour l'essentiel disponibles, bien qu'une certaine adaptation soit nécessaire. L'objectif est alors d'écrire en Caml la partie haute du programme mettant en œuvre la méthode de décomposition de domaines afin d'évaluer l'intérêt d'utiliser un langage fonctionnel dans l'écriture de tels programmes de calcul scientifique, en particulier quant à la possibilité de généralisation, avec à terme, l'écriture d'un générateur automatique d'interfaces de couplage.

Le travail comporte un important aspect de spécifications d'interfaces entre programmes, il importe donc d'utiliser les produits les plus modernes et les plus performants de la recherche en matière de génie logiciel. C'est pourquoi on se propose d'utiliser le langage Caml pour concevoir l'architecture globale du système. OCaml a l'avantage de fournir des outils de spécification divers, puissants et de haut niveau : fonctions, signatures (types des modules), structures (implémentations des modules), foncteurs (fonctions des modules dans les modules), classes et objets. On utilisera les moyens les plus appropriés pour décrire :

- le système complet,
- le couplage des solveurs de sous-domaines :
  - par la spécification des interfaces entre modules-solveurs,
  - par l'étude de l'interfaçage des solveurs, généralement écrits en C ou Fortran, pour qu'ils répondent à la spécification Caml précédente.

L'aspect exécution en parallèle des différents codes à coupler n'était pas, a priori, un objectif prioritaire de la présente étude. Cependant, d'une part, les techniques numériques de couplage de codes et de parallélisation d'un code (couplage avec lui-même) sont similaires, et d'autre part, l'environnement de développement OCamlP3l (écrit en OCaml) intègre la gestion de processus concurrents. C'est pourquoi nous choisissons comme problème modèle la résolution de l'équation de Poisson (problème elliptique type) en trois dimensions d'espace par décomposition de domaine.

Pour la réalisation d'une première maquette, l'idée directrice est la rapidité de la mise en œuvre. Nous simplifions donc encore le problème modèle au cas bi-dimensionnel sur une grille régulière, les sous-domaines correspondent à une sous-grille régulière et les conditions de transmission sont de type Dirichlet.

2 Un problème modèle : le Laplacien 3D

Nous allons décrire dans cette section l'algorithme de couplage 3D pour une expérience donnée. Supposons donc pour simplifier que nous voulons résoudre le problème modèle suivant :

$$\left\{ \begin{array}{rl} -\Delta p = f & \text{sur } \Omega \subset \mathbb{R}^3 \\ p = 0 & \text{sur } \Gamma = \partial\Omega \end{array} \right. \tag{1}$$

où Ω est un ouvert de $\mathbb{R}^3$ et p et f des fonctions de $\mathbb{R}^3$ dans $\mathbb{R}$.

On va résoudre ce problème en découpant le domaine Ω en sous-domaines sans recouvrement et le couplage sera obtenu à l'aide de conditions de transmission mixtes de type Robin. L'algorithme de couplage consiste à résoudre un problème de point fixe de type :

$$F(\lambda) = \lambda$$

où λ est la valeur de la fonction p sur l'ensemble de toutes les interfaces entre tous les sous-domaines. On montre que ce problème de point fixe se ramène à la résolution d'un système linéaire

$$A(\lambda) = b$$

que l'on résoud de façon itérative, par exemple à l'aide de l'algorithme du Gradient Bi-Conjugué Stabilisé (Bi-CGStab). Cette technique de couplage sera appliquée à un code d'écoulement et de transport en milieu poreux appelé LifeV.

2.1 Décomposition de domaine sans recouvrement

Présentons d'abord quelques définitions et notations utiles pour la suite. Soit Ω un domaine de $\mathbb{R}^3$. Ce domaine est supposé partitionné en n sous-domaines disjoints, soit, avec $I = \{1, 2, \ldots, n\}$,

$$\bar{\Omega} = \bigsqcup_{i \in I} \bar{\Omega}_i.$$

Deux sous-domaines Ω_i et Ω_j sont dits **voisins** si leurs frontières ont une intersection de dimension 2. Pour $i \in I$, le nombre de voisins du sous-domaine Ω_i est noté n_i et l'ensemble des indices de ces voisins est noté $\mathcal{V}_i = \{j_1, j_2, \ldots, j_{n_i}\}$. Les ensembles d'indices I et $\mathcal{V}_i$ sont munis de l'ordre des entiers naturels.

Pour $j \in \mathcal{V}_i$, l'interface géométrique $\partial\Omega_i \cap \partial\Omega_j$ entre les deux voisins est notée Σ_{ij} ou Σ_{ji} suivant qu'elle est vue du sous-domaine Ω_i ou Ω_j (voir Figure 1).

On appelle **bord intérieur du sous-domaine** Ω_i la réunion de ses interfaces vues de lui-même. Cette réunion est notée Σ_i, elle est ordonnée par $\mathcal{V}_i$. Le reste du bord (éventuellement vide) est noté Γ_i. Ainsi,

$$\Sigma_i = \bigsqcup_{j \in \mathcal{V}_i} \Sigma_{ij}, \quad \partial\Omega_i = \Gamma_i \sqcup \Sigma_i \quad \text{et} \quad \Gamma = \partial\Omega = \bigsqcup_{i \in I} \Gamma_i.$$

L'ensemble des couples de voisins est noté $\mathcal{V}$. Il est muni de l'ordre lexicographique sur $\mathbb{N}^2$. On note s la permutation involutive réalisant la symétrie de toutes les interfaces,

$$\forall (i,j) \in \mathcal{V}, \quad s(i,j) = (j,i) \in \mathcal{V}.$$

On appelle **structure intérieure du domaine** Ω la réunion de tous les bords intérieurs, c'est-à-dire la réunion de toutes les interfaces, vues de l'un ou de l'autre de deux voisins (chaque interface géométrique étant dédoublée). Cette réunion est notée Σ. Elle est ordonnée par $\mathcal{V}$,

$$\Sigma = \bigsqcup_{i \in I} \Sigma_i = \bigsqcup_{i \in I} \bigsqcup_{j \in \mathcal{V}_i} \Sigma_{ij} = \bigsqcup_{(i,j) \in \mathcal{V}} \Sigma_{ij}.$$

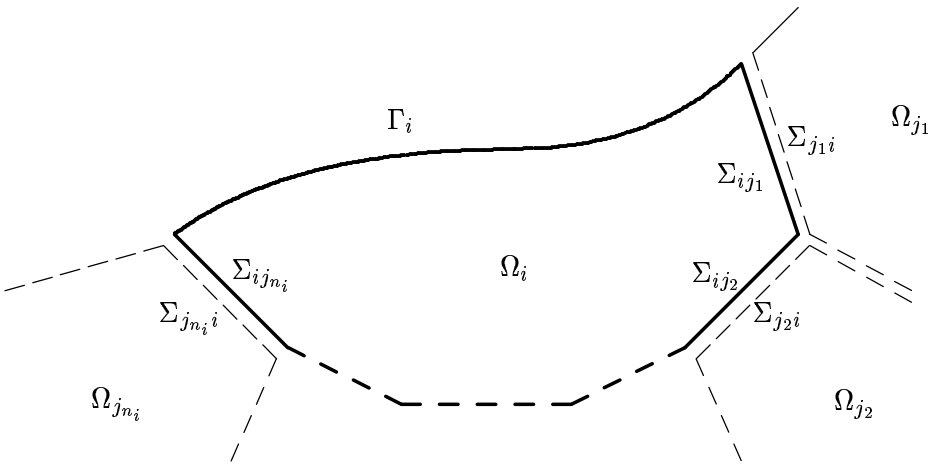

FIG. 1 – *Le sous-domaine Ω_i, ses voisins et les différentes interfaces (dans le plan).*

Pour $(i,j) \in \mathcal{V}$, on pose $\tilde{\Sigma}_{ij} = \Sigma_{ji}$. On appelle **bord extérieur du sous-domaine** Ω_i la réunion de ses interfaces vues de ses voisins. Cette réunion est notée $\tilde{\Sigma}_i$. Elle est également ordonnée par $\mathcal{V}_i$,

$$\tilde{\Sigma}_i = \bigsqcup_{j \in \mathcal{V}_i} \tilde{\Sigma}_{ij} = \bigsqcup_{j \in \mathcal{V}_i} \Sigma_{ji}.$$

De la même façon, on appelle **structure extérieure du domaine** Ω la réunion de tous les bords extérieurs et on la note $\tilde{\Sigma}$. C'est aussi la réunion de toutes les interfaces, vues de l'un ou de l'autre de deux voisins, mais cette fois-ci ordonnée par $s(\mathcal{V})$,

$$\tilde{\Sigma} = \bigsqcup_{i \in I} \tilde{\Sigma}_i = \bigsqcup_{i \in I} \bigsqcup_{j \in \mathcal{V}_i} \tilde{\Sigma}_{ij} = \bigsqcup_{i \in I} \bigsqcup_{j \in \mathcal{V}_i} \Sigma_{ji} = \bigsqcup_{(i,j) \in s(\mathcal{V})} \Sigma_{ij}.$$

Chaque sous-domaine Ω_i est supposé maillé, et est identifié a son maillage. Chaque interface Σ_{ij} est supposée munie de la trace du maillage Ω_i, et est également identifiée à ce maillage-trace. Ainsi, les maillages-traces des interfaces vues de l'un ou de l'autre de deux voisins peuvent ne pas coïncider. Il faut bien noter que l'ensemble des sommets/nœuds du maillage de peau Σ_{ij} est un sous-ensemble de l'ensemble des sommets/nœuds du maillage Ω_i, mais que la numérotation est différente. Il est donc nécessaire de construire une table de correspondance entre les deux maillages.

2.2 Technique de couplage avec des conditions de type Robin

Pour cet exemple, la difficulté réside dans le fait que les maillages des différents sous-domaines ne se raccordent pas.

Étant donnés deux sous-domaines voisins Ω_i et Ω_j (c'est-à-dire avec $(i,j) \in \mathcal{V}$), les restrictions respectives p_i et p_j de la solution du problème (1) vérifient les conditions de raccord à l'interface

$$p_i - p_j = 0 \quad \text{et} \quad \frac{\partial p_i}{\partial \nu_i} + \frac{\partial p_j}{\partial \nu_j} = 0 \quad \text{sur } \partial\Omega_i \cap \partial\Omega_j \tag{2}$$

où ν est la normale extérieure. Pour α_{ij} et α_{ji} réels positifs, (2) est équivalent à

$$\frac{\partial p_i}{\partial \nu_i} + \alpha_{ij} p_i = -\frac{\partial p_j}{\partial \nu_j} + \alpha_{ij} p_j \quad \text{et} \quad \frac{\partial p_i}{\partial \nu_i} - \alpha_{ji} p_i = -\frac{\partial p_j}{\partial \nu_j} - \alpha_{ji} p_j \quad \text{sur } \partial\Omega_i \cap \partial\Omega_j,$$

c'est-à-dire aux conditions mixtes de type Robin

$$(3)\qquad \begin{cases} \dfrac{\partial p_i}{\partial \nu_i} + \alpha_{ij} p_i = -\dfrac{\partial p_j}{\partial \nu_j} + \alpha_{ij} p_j \\ \dfrac{\partial p_j}{\partial \nu_j} + \alpha_{ji} p_j = -\dfrac{\partial p_i}{\partial \nu_i} + \alpha_{ji} p_i \end{cases} \quad \text{sur } \partial\Omega_i \cap \partial\Omega_j.$$

Chaque restriction p_i est donc solution du problème suivant :

$$(4)\qquad \begin{cases} -\Delta p_i = f_i & \text{sur } \Omega_i \\ p_i = 0 & \text{sur } \Gamma_i \\ \dfrac{\partial p_i}{\partial \nu_i} + \alpha_{ij} p_i = -\dfrac{\partial p_j}{\partial \nu_j} + \alpha_{ij} p_j & \text{sur } \Sigma_{ij}, \text{ pour } j \in \mathcal{V}_i. \end{cases}$$

Pour chaque sous-domaine Ω_i, étant donnés α_i et β_i positifs (constants sur chaque interface), nous considérons donc l'opérateur bilinéaire de type Robin-Robin

$$(5)\qquad S_i : (\lambda_i \text{ sur } \Sigma_i, f_i \text{ sur } \Omega_i) \longmapsto \mu_i = -\frac{\partial p_i}{\partial \nu_i} + \beta_i p_i \quad \text{sur } \Sigma_i$$

où p_i est solution du problème

$$(6)\qquad \begin{cases} -\Delta p_i = f_i & \text{sur } \Omega_i \\ p_i = 0 & \text{sur } \Gamma_i \\ \dfrac{\partial p_i}{\partial \nu_i} + \alpha_i p_i = \lambda_i & \text{sur } \Sigma_i. \end{cases}$$

Pour chaque couple de voisins $(i,j) \in \mathcal{V}$, soit l'opérateur de projection L^2

$$(7)\qquad P_{i\to j} : \lambda_{ij} \text{ sur } \Sigma_{ij} \longmapsto \tilde{\lambda}_{ij} \text{ sur } \tilde{\Sigma}_{ij},$$

et soit l'opérateur de restriction

$$(8)\qquad R_{ij} : \lambda_i = \begin{pmatrix} \lambda_{ij_1} \\ \lambda_{ij_2} \\ \vdots \\ \lambda_{ij_{n_i}} \end{pmatrix} \text{ sur } \Sigma_i \longmapsto \lambda_{ij} \text{ sur } \Sigma_{ij}.$$

Soient également

$$(9)\qquad P_i : \lambda_i \text{ sur } \Sigma_i \longmapsto \tilde{\lambda}_i = \begin{pmatrix} \tilde{\lambda}_{ij_1} \\ \tilde{\lambda}_{ij_2} \\ \vdots \\ \tilde{\lambda}_{ij_{n_i}} \end{pmatrix} \text{ sur } \tilde{\Sigma}_i,$$

$$(10)\qquad R_i : \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix} \text{ sur } \Sigma \longmapsto \lambda_i \text{ sur } \Sigma_i$$

et

$$(11)\qquad \tilde{R}_{ij} : \tilde{\lambda}_i \text{ sur } \tilde{\Sigma}_i \longmapsto \tilde{\lambda}_{ij} \text{ sur } \tilde{\Sigma}_{ij}.$$

Pour tout $(i,j) \in \mathcal{V}$, nous avons les propriétés équivalentes suivantes :

$$(12)\qquad P_{i\to j} R_{ij} = \tilde{R}_{ij} P_i, \quad P_i = \sum_{j\in\mathcal{V}_i} \tilde{R}_{ji}^T P_{i\to j} R_{ij} \quad \text{et} \quad P_{i\to j} = \tilde{R}_{ij} P_i R_{ij}^T.$$

Nous définissons aussi les opérateurs globaux

$$S : \left(\lambda \text{ sur } \Sigma, f = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \text{ sur } \Omega \right) \longmapsto \mu = \begin{pmatrix} S_1(\lambda_1, f_1) \\ S_2(\lambda_2, f_2) \\ \vdots \\ S_n(\lambda_n, f_n) \end{pmatrix} \text{ sur } \Sigma \tag{13}$$

et

$$P : \lambda \text{ sur } \Sigma \longmapsto \tilde{\lambda} = \begin{pmatrix} \tilde{\lambda}_1 \\ \tilde{\lambda}_2 \\ \vdots \\ \tilde{\lambda}_n \end{pmatrix} \text{ sur } \tilde{\Sigma}. \tag{14}$$

La solution du problème (1), ou bien des problèmes (4) pour $i \in I$, doit satisfaire le problème de point fixe suivant (avec $\beta_{ij} = \alpha_{ji}$) :

$$s \circ P \circ S(\lambda, f) = \lambda. \tag{15}$$

C'est-à-dire, pour $(i,j) \in \mathcal{V}$,

$$\lambda_{ij} = \tilde{\mu}_{ji} = P_{j \to i} \mu_{ji} = P_{j \to i} R_{ji} \mu_j = P_{j \to i} R_{ji} S_j(\lambda_j, f_j) = \tilde{R}_{ji} P_j S_j(\lambda_j, f_j). \tag{16}$$

Problème mis sous la forme d'un système linéaire

En posant $b_{ij} = \tilde{R}_{ji} P_j S_j(0, f_j)$ et $T_{ij} = \tilde{R}_{ji} P_j S_j(., 0)$, le problème de point fixe (15) est équivalent à la résolution du système linéaire

$$A\lambda = b \tag{17}$$

où A est la matrice carrée ayant la structure bloc suivante :

$$\begin{cases} A_{ii} = \mathrm{Id}_{n_i} & \\ A_{ij} = -T_{ji} & \text{si } (i,j) \in \mathcal{V} \\ A_{ij} = 0 & \text{sinon,} \end{cases} \tag{18}$$

et b la collection

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \text{avec} \quad b_i = \begin{pmatrix} b_{ij_1} \\ b_{ij_2} \\ \vdots \\ b_{ij_{n_i}} \end{pmatrix}. \tag{19}$$

Attention, dans le bloc A_{ij}, seule la j^{e} ligne est non nulle. En fait, nous avons

$$(A\lambda)_{ij} = \lambda_{ij} - T_{ij} \lambda_j. \tag{20}$$

Algorithme pour le produit matrice-vecteur

L'algorithme pour le calcul du produit matrice-vecteur $(\lambda \mapsto (A\lambda))$ est le suivant :

Pour $i \in I$
  $\lambda_i = R_i \lambda \qquad (\Sigma \to \Sigma_i)$
  Pour $j \in \mathcal{V}_i$
    $\lambda_{ij} = R_{ij} \lambda_i \qquad (\Sigma_i \to \Sigma_{ij})$

Pour $j \in I$

$\mu_j = S_j(\lambda_j, 0) \qquad (\Sigma_j \to \Sigma_j)$
Pour $i \in \mathcal{V}_j$
$\mu_{ji} = R_{ji}\mu_j \qquad (\Sigma_j \to \Sigma_{ji})$
$\tilde{\mu}_{ji} = P_{j\to i}\mu_{ji} \qquad (\Sigma_{ji} \to \tilde{\Sigma}_{ji} = \Sigma_{ij})$
$(A\lambda)_{ij} = \lambda_{ij} - \tilde{\mu}_{ji} \qquad (\Sigma_{ij} \to \Sigma_{ij})$

$(A\lambda) = 0$
Pour $i \in I$
$(A\lambda)_i = 0$
Pour $j \in \mathcal{V}_i$
$(A\lambda)_i = (A\lambda)_i + R_{ij}^T(A\lambda)_{ij} \qquad (\Sigma_{ij} \to \Sigma_i)$
$(A\lambda) = (A\lambda) + R_i^T(A\lambda)_i \qquad (\Sigma_i \to \Sigma)$

Les codes à coupler correspondent aux opérateurs de type Robin-Robin S_i. Il faut donc connaître :

- la table de connectivité des sous-domaines (= graphe de voisinage = $\mathcal{V}$),
- les maillages de chaque interface (pour les opérateurs de projection $P_{i\to j}$).

2.3 Algorithme du Gradient bi-Conjugué Stabilisé (Bi-CGStab)

Pour la résolution du système linéaire (17), nous utilisons l'algorithme itératif Bi-CGStab, qui est adapté à des matrices non symétriques. Voici le détail de cet algorithme :

x_0 donné
$r_0 = b - Ax_0$
$\widehat{r}_0$ tel que $(\widehat{r}_0, r_0) \neq 0$, par exemple $\widehat{r}_0 = r_0$
$\rho_0 = \alpha_0 = \omega_0 = 1$
$p_0 = v_0 = 0$
pour $i \geq 1$ **faire**
$\rho_i = (\widehat{r}_0, r_{i-1})$
$\beta = \dfrac{\rho_i}{\rho_{i-1}}\dfrac{\alpha_{i-1}}{\omega_{i-1}}$
$p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$
$v_i = Ap_i \longleftarrow$ **appel au calcul sur chaque sous-domaine** Ω_i
$\alpha_i = \dfrac{\rho_i}{(\widehat{r}_0, v_i)}$
$s = r_{i-1} - \alpha_i v_i$
$t = As \longleftarrow$ **appel au calcul sur chaque sous-domaine** Ω_i
$\omega_i = \dfrac{(t,s)}{(t,t)}$
$x_i = x_{i-1} + \alpha_i p_i + \omega_i s$
$r_i = s - \omega_i t$
si convergence **alors** s'arrêter
fait

Cet algorithme va être écrit en OCaml avec les nouveaux termes définis par OCamlP3l (cf. section 4).

2.4 Opérateur de projection $L^2(\Omega)$

Dans le paragraphe précédent l'alogrithme de couplage emploie des projecteurs $P_{i\to j}$ d'un sous-domaine Ω_i sur un sous-domaine Ω_j. Cet opérateur permet de transformer m_i valeurs d'une fonction

scalaire discrétisée sur le bord Σ_{ij} en m_j valeurs sur le bord Σ_{ji} et établir ainsi une correspondance entre Ω_i et Ω_j (de maillage différent, voire de dimension différente). L'algorithme de projection pour des domaines 3D/2D que nous utiliserons par la suite repose sur la définition suivante. On peut définir la projection $L^2(\Omega)$ sur l'espace $\mathcal{P}_0(\mathcal{T}_{h_l})$ des fonctions constantes par maille sur un maillage $\mathcal{T}_{h_l}$ comme suit, pour tout $x \in \Omega$, pour une fonction $f \in L^2(\Omega)$:

$$P(f)(x) = \sum_{\substack{\mathcal{C}\in\mathcal{T}_{h_l} \\ x\in\mathcal{C}}} \frac{\chi_{\mathcal{C}}(x)}{\mathrm{mes}(\mathcal{C})} \int_{\mathcal{C}} f(y)dy \tag{21}$$

où $\chi_{\mathcal{C}}$ est la fonction indicatrice de $\mathcal{C}$. On remarque que cette formule est similaire à la construction d'une solution approchée par volumes finis.

Si l'on récrit cette projection pour deux domaines Ω_1 et Ω_2 ayant une interface commune et pour des solutions approchées $v_1 \in P_0(\mathcal{T}_1)$ (resp. $v_2 \in P_0(\mathcal{T}_2)$) constantes par morceau sur Ω_1 (resp. Ω_2), on obtient

$$\begin{aligned} P_{1\to2}(v_1)_{\mathcal{D}} &= \sum_{\substack{\mathcal{C}\in\mathcal{T}_{h_1}, \\ \mathcal{D}\cap\mathcal{C}\neq\emptyset}} \mu^{12}_{\mathcal{C},\mathcal{D}}\, v_{1\,\mathcal{C}} \\ P_{2\to1}(v_2)_{\mathcal{C}} &= \sum_{\substack{\mathcal{D}\in\mathcal{T}_{h_2}, \\ \mathcal{D}\cap\mathcal{C}\neq\emptyset}} \mu^{21}_{\mathcal{D},\mathcal{C}}\, v_{2\,\mathcal{D}} \end{aligned} \tag{22}$$

où

$$\mu^{12}_{\mathcal{C},\mathcal{D}} = \frac{\mathrm{mes}(\mathcal{D}\cap\mathcal{C})}{\mathrm{mes}(\mathcal{D})} \quad \text{et} \quad \mu^{21}_{\mathcal{D},\mathcal{C}} = \frac{\mathrm{mes}(\mathcal{D}\cap\mathcal{C})}{\mathrm{mes}(\mathcal{C})}. \tag{23}$$

On forme ainsi une combinaison convexe, en effet :

$$\sum_{\mathcal{C}\in\mathcal{T}_{h_1}} \mathrm{mes}(\mathcal{D}\cap\mathcal{C}) = \mathrm{mes}(\mathcal{D}).$$

Il est à noter que la projection ainsi définie vérifie une propriété importante, concernant la conservation de la masse :

$$\int_{\Omega_{12}} \sum_{\mathcal{D}\in\mathcal{T}_{h_2}} P_{1\to2}(v_1)\chi_{\mathcal{D}}(x)dx = \int_{\Omega_{12}} \sum_{\mathcal{C}\in\mathcal{T}_{h_1}} v_{1\,\mathcal{C}}\chi_{\mathcal{C}}(x).$$

On dit alors qu'elle est conservative.

Ainsi pour deux domaines, en partant de solutions approchées à un instant t^n donné, on utilisera deux schémas numériques séparément puis on effectuera la correction par le couplage en évaluant les projections (projection $P_{1\to2}$ et $P_{2\to1}$), pour obtenir les solutions à l'instant suivant t^{n+1}. Actuellement le procédé de projection d'un maillage quelconque sur un autre a été écrit en Matlab par Mancip, cf. [3], et doit être récrit en une fonction C que nous interfacerons pour être intégré au coupleur. Comme on peut le voir avec (23), le calcul de projection se ramène au calcul de l'intersection de mailles, c'est-à-dire de parties convexes du plan ou de l'espace. La routine que nous voulons interfacer réalise ce calcul d'intersection et repose sur le principe que l'intersection de parties convexes reste une partie convexe.

2.5 Le programme LifeV

Le couplage que nous voulons réaliser en 3D (avec interfaces 2D) consiste à coupler avec lui-même un code modélisant l'écoulement et le transport en milieu poreux. Le couplage sur plusieurs domaines rendra compte de la modélisation pour un milieu géologique hétérogène où chaque domaine représente une couche géologique particulière.

Le code que nous utilisons est LifeV, un code général d'éléments finis qui possède plusieurs modules résolvant différents type d'équations différentielles et dont le module Darcy résout le système suivant sur un domaine Ω :

$$\begin{cases} \text{div} u = f & \text{sur } \Omega \\ u = -K\vec{\nabla}p & \text{sur } \Omega \\ p = p_{\text{Dirichlet}} & \text{sur } \Gamma_D \\ u.\nu = q_{\text{Neuman}} & \text{sur } \Gamma_N \end{cases}$$

Le module Darcy a été écrit en C++ par Martin, cf. [4], en collaboration avec différentes équipes dont le projet Bang ; pour faire de la décomposition de domaine, le programme a été modifié pour transmettre les conditions de type Robin sur l'interface, à savoir bien définir l'opérateur de type Robin-Robin (5).

3 Un problème modèle simplifié : le Laplacien 2D sur grille régulière

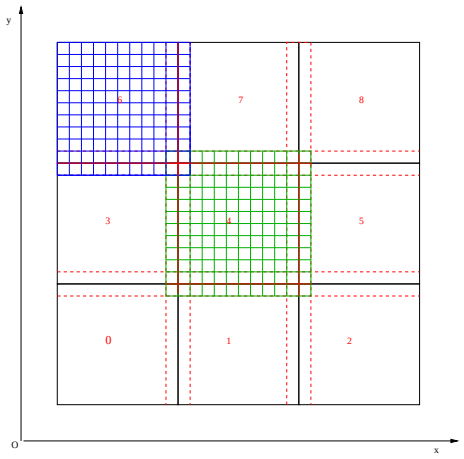

FIG. 2 – *Le domaine Ω subdivisé en 9 sous-domaines.*

Pour mettre en évidence l'architecture du coupleur, (cf. section 4 : utilisation de nouveaux modules fournis par OcamP3l) nous avons simplifié le problème pour obtenir un programme court (170 lignes), et facile à mettre au point. La simplification porte sur plusieurs points :

– On part du même système différentiel mais on se place en dimension 2,

$$\begin{cases} -\Delta p = f & \text{sur } \Omega \subset \mathbb{R}^2 \\ p = 0 & \text{sur } \Gamma = \partial\Omega. \end{cases} \tag{24}$$

– Le domaine Ω est un rectangle et on le décompose en sous-domaines rectangulaires Ω_i se recouvrant d'un certain nombre de mailles. Sur l'exemple de la figure 2 on a partagé Ω en 9 sous-domaines et on a représenté le maillage des domaines 4 et 6.

– Le maillage de chaque domaine est régulier (c'est un quadrillage régulier dont la taille de maille est $(\Delta x, \Delta y)$). On n'a pas besoin de stocker le maillage en mémoire, il suffit de connaître le nombre N de points intérieurs et la taille du domaine (ce qui ne sera pas le cas par la suite pour un maillage non structuré que le coupleur devra éventuellement lire selon un format d'échange prédéfini).

– Les maillages entre chaque sous-domaine sont concordants: la projection $P_{i\to j}$ d'un domaine Ω_i sur un domaine Ω_j voisin est donc l'identité.

– Le couplage se fait en résolvant sur chaque sous-domaine Ω_i le problème suivant :

$$\left\{ \begin{array}{rl} \Delta u = f & \text{sur } \Omega_i \\ u = 0 & \text{sur } \Gamma_i \\ u = \lambda_i & \text{sur } \Sigma_i \end{array} \right. \tag{25}$$

où les λ_i sont des conditions de type Dirichlet que nous allons définir par la suite.

3.1 Décomposition de domaine avec recouvrement

Nous donnons quelques définitions d'indices :

$$\left\{ \begin{array}{rcl} N_x & = & \text{nombre de domaine selon } Ox \\ m_x & = & \text{nombre de maille par domaine selon } Ox \\ n_x & = & \text{nombre de maille total selon } Ox \\ p_x & = & \text{nombre de maille de recouvrement selon } Ox. \end{array} \right.$$

Mêmes définitions pour m_y, N_y, n_y, p_y selon l'axe Oy.

Un point du maillage de Ω est repéré par ses coordonnées

$$\left\{ \begin{array}{l} x_i = x_0 + i\Delta x, \quad i = 0, \ldots, N_x \\ y_j = y_0 + j\Delta y, \quad j = 0, \ldots, N_y \end{array} \right.$$

où (x_0, y_0) est le coin inférieur gauche de Ω. La décomposition s'appuie sur des sous-grilles régulières repérées par leur coin inférieur gauche (x_I, y_J) avec $x_I = x_0 + Im_x\Delta x$ et $y_J = y_0 + Jm_y\Delta y$. Chaque sous-domaine Ω_i est repéré par son numéro $i = I + N_xJ$ (avec $0 \leq I \leq N_x - 1$ et $0 \leq J \leq N_y - 1$) et l'ensemble des points de son maillage correspond aux (x_i, y_j) tels que :

$$\left\{ \begin{array}{l} i = \max(Im_x - p_x, 0), \ldots, \min((I+1)m_x + p_x, n_x) \\ j = \max(Jm_y - p_y, 0), \ldots, \min((J+1)m_y + p_y, n_y). \end{array} \right.$$

La connectivité entre les sous-domaines est simple, chaque Ω_i a au plus quatre voisins :

$$\left\{ \begin{array}{l} \text{Est : numéro} = i + 1 \\ \text{Ouest : numéro} = i - 1 \\ \text{Nord : numéro} = i + N_x \\ \text{Sud : numéro} = i - N_x. \end{array} \right.$$

3.2 Technique de couplage avec des conditions de type Dirichlet

Les conditions de transmission consistent à communiquer à ses voisins les conditions de Dirichlet dont il a besoin (cf. figure 3). Par exemple, le sous-domaine $I + N_xJ$ communique à son voisin Est (si $I < N_x - 1$) les valeurs en (x_i, y_j) pour $i = (I+1)m_x - p_x$ et $J.m_y - p_y \leq j \leq (J+1)m_y + p_y$.

3.3 Mise en œuvre : schémas implicite à 5 points

Pour résoudre le Laplacien, nous avons utilisé une méthode implicite définie par le schémas suivant à 5 points :

$$U^{n+1}_{i+1,j} = \omega(U^n_{i+1,j} + U^n_{i,j+1} + U^{n+1}_{i-1,j} + U^{n+1}_{i,j-1} - \Delta x\Delta y F_{i,j})/4 + (1-\omega)U^n_{i,j}, \tag{26}$$

où $F_{i,j} = f(x_i, y_j)$ et $\omega = \frac{2}{1+\sqrt{(1-\eta^2)}}$ est le coefficient de relaxation optimal (η étant le rayon spectral de la matrice associée).

Sur le carré unité, $\Omega = [0,1] \times [0,1]$, nous choisissons $f(x,y) = 2(y^2 - y + x^2 - x)$, donc la solution exacte est $u(x,y) = -xy(1-x)(1-y)$.

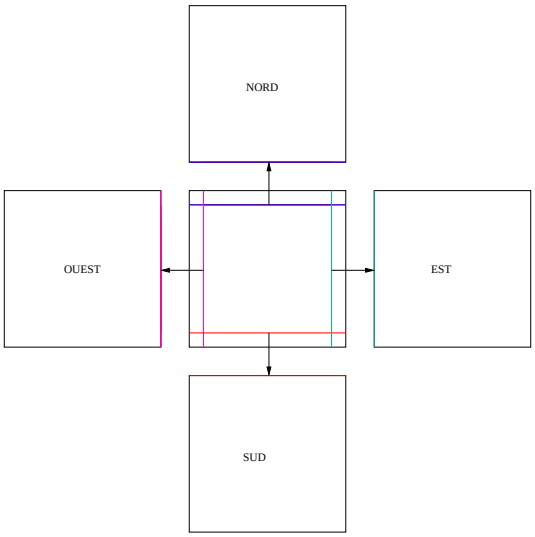

FIG. 3 – *Communications entre voisins.*

4 L'environnement de programmation OCamlP3L

4.1 Présentation

Le projet Cristal contribue dans le cadre de l'ACI GRID « CARAML », depuis l'an dernier, au développement de bibliothèques pour le calcul haute-performance et globalisé en Objective Caml. L'action réunit des chercheurs impliqués dans le parallélisme de données (laboratoires LIFO à Orléans, PPS à Paris 7, LACL à Paris 12), dans les calculs concurrents (projet Moscova), ainsi qu'une expertise en matière de typage et du langage Objective Caml au travers du projet Cristal. L'action s'appuie aussi sur l'activité OCamlP3L dans laquelle R. Di Cosmo et X. Leroy ont été impliqués en 1998, en collaboration avec l'Université de Pise.

C'est sur le conseil de P. Weis et de R. Di Cosmo que nous avons utilisé pour la programmation du coupleur des modules (Seqp3l, Parp3l, Grap3l) écrits en OCaml et regroupé sous le terme d'OCamlP3l, cf. [2]. OCamlP3l est une librairie qui rend disponible au programmeur un ensemble de squelettes de parallélisme de haut niveau, ce qui permet de décrire très concisément la structure parallèle d'un programme, dans l'esprit de [1]. L'objectif des modules d'OCamlP3l est d'automatiser la parallélisation d'une application développée en OCaml. Bien que notre objectif intial soit de coupler des applications (écrites en C, C++ ou Fortran) et non nécessairement de les paralléliser, nous avons pu utiliser les nouveaux mots clés définis dans ces modules.

L'intérêt de ces modules est donc double. D'une part, il permet de faire communiquer aisément différents processus à exécuter en même temps. Nous n'avons pas à nous préoccuper de la création ni des communications explicites entre les processus, nous n'avons pas à nous occuper de la répartition partagée de la mémoire, tout ceci est géré à un niveau bien évidemment plus bas de programmation. D'autre part, ces modules permettent de paralléliser certaines parties des programmes à coupler. Le programme utilise des "squelettes" (qui sont en fait des fonctions du second ordre) pour décrire la structure de l'application "coupleur". Il faut exprimer les portions séquentielles de l'application à paralléliser comme paramètres du squelette.

Ensuite, OCamlP3l permet de choisir, sans modifier le programme source, une implémentation séquentielle des squelettes, qui permet une mise au point de l'algorithme de l'utilisateur, ou une implementation parallèle, qui permet le déploiement sur une grille de calculateurs.

4.2 Squelettes de base

Nous rappellons ici les squelettes OCamlP3l utilisés dans le coupleur, en donnant les définitions en OCaml qui correspondent à leur implémentation séquentielle.

Exécution parallèle d'un calcul sur les composantes d'un vecteur

```
* let mapvector (f, n) = (Array.map f);;
```

Le squelette d'exécution parallèle sur un vecteur utilise la fonction `mapvector`. Il permet d'encapsuler l'exécution d'un squelette `f` sur toutes les composantes d'un flot vectoriel de données sur `n` processus parallèles. Si `f` a pour type OCaml `'a stream -> 'b stream` et `n` est un entier, alors `mapvector (f, n)` a pour type `'a vector stream -> 'b vector stream`.

Boucle

```
* let rec loop (c, f) x =
    let r = f x in
    if c r then loop (c, f) r else r;;
```

Le squelette de boucle utilise la fonction récursive `loop`. Il permet d'encapsuler l'exécution répétée d'un squelette `f` sur un flot de données jusqu'à ce que la condition `c` soit fausse. Si `f` a pour type OCaml `'a -> 'a` et `c` a pour type `'a -> bool`, alors `loop (c, f)` a pour type `'a stream -> 'a stream`.

Composition séquentielle

```
* let (|||) f g = fun x -> g (f x);;
```

Le squelette de composition séquentielle utilise l'opérateur infixe `|||` (ou la fonction `(|||)`). Il permet d'encapsuler l'exécution séquentielle des deux squelettes `f` et `g` sur un flot de données. Si `f` a pour type OCaml `'a stream -> 'b stream` et `g` a pour type `'b stream -> 'c stream`, alors `f ||| g` a pour type `'a stream -> 'c stream` (`f ||| g` signifie donc g∘f).

Exécution séquentielle

```
* let seq f = f;;
```

Le squelette d'exécution séquentielle utilise la fonction `seq`. Il permet d'encapsuler l'exécution de la fonction `f` dans un processus séquentiel et de l'appliquer à un flot de données. Si `f` a pour type OCaml `'a -> 'b`, alors `seq f` a pour type `'a stream -> 'b stream`.

Encapsulation externe

```
* let startstop (f1, init1) (f2, init2, finalize) expr =
    init1 (); init2 ();
    try
      while true do f2 (expr (f1 ())) done
    with End_of_file -> finalize ();;
```

La fonction d'encapsulation externe utilise la fonction `startstop`. Elle permet d'encapsuler le squelette `expr` en générant les flots de données d'entrée (par `init1` et `f1`) et en affichant les résultats (par `init2`, `f2` et `finalize`). Elle crée un réseau OCamlP3l.

Exécution parallèle d'un réseau

```
* let pardo expr = expr ();;
```

La fonction d'exécution parallèle d'un réseau utilise la fonction `pardo`. Elle permet d'exécuter en parallèle un réseau OCamlP3l défini par un appel à la fonction `startstop`.

4.3 Compilation

Pour la compilation d'un réseau on utilise le compilateur `ocampl3lcc` avec l'une des trois options `-seq`, `-par` ou `-gra` suivant que l'on désire exécuter le réseau en mode séquentiel ou parallèle, ou bien visualiser graphiquement sa structure. Il est également possible de compiler directement avec le compilateur OCaml en lui passant l'option `-custom` et les modules compilés `seqp3l.cmo`, `parp3l.cmo` et `grap3l.cmo`.

La compilation d'un réseau dont le source OCaml se trouve dans le fichier `reseau.ml` produit des exécutables `reseau.seq` ou `reseau.par`. L'exécution séquentielle s'obtient simplement par la commande `reseau.seq`. L'exécution parallèle s'obtient en lançant d'abord l'exécutable `reseau.par` en arrière-plan sur chaque nœud, puis par la commande `reseau.par -p3lroot machine1 machine2 ...machineN`.

5 Tests numériques

5.1 Mise en œuvre

Programme OCaml : `coupleur.ml`

Voici la partie principale du coupleur définie à partir de la fonction `startstop`, fonction à trois arguments :

- Le premier nous servira à initialiser le couplage en générant des conditions de bord pour chaque sous-domaine. Par la suite il faudra lire dans un fichier, durant cette étape, les informations telles la table de connectivité entre sous-domaines Ω_i ainsi que la nature et la structure des maillages pour chaque sous-domaine.
- Le deuxième permet de "finaliser" l'ensemble des calculs fait sur les différents processus.
- Enfin le troisième argument permet de décrire véritablement (même si ici on a simplifié) la structure du couplage ; ici on a une boucle qui à chaque itération enchaîne grace à la composition séquentielle `|||` le calcul du Laplacien sur chaque sous-domaine, la projection des nouveaux vecteurs "bord" et la visualisation des résultats à chaque itération de calcul sous `GNUplot`.

```
let program () =
  startstop
    (generate_input_stream, ignore)
    (print_result, ignore, ignore)
    (loop (convergence, mapvector (seq (calcul_sous_domaine), nbproc)
      ||| seq (proj) ||| seq (plot)))
in
pardo program;;
```

Programme C : `poisson.c`

Une fois que les processus fils sont lancés et que leur entrée/sortie standard a été redirigée vers celle du coupleur il faut que ceux-ci restent en attente de lecture et d'écriture. Nous utilisons donc une boucle "infinie" et le programme C a la structure suivante :

```
main () {
  while (1) do {
    /* lecture du vecteur bord sur Stdin   */
    /* résolution du Laplacien             */
    /* écriture du vecteur bord sur Stdout */
  }
}
```

5.2 Résultats

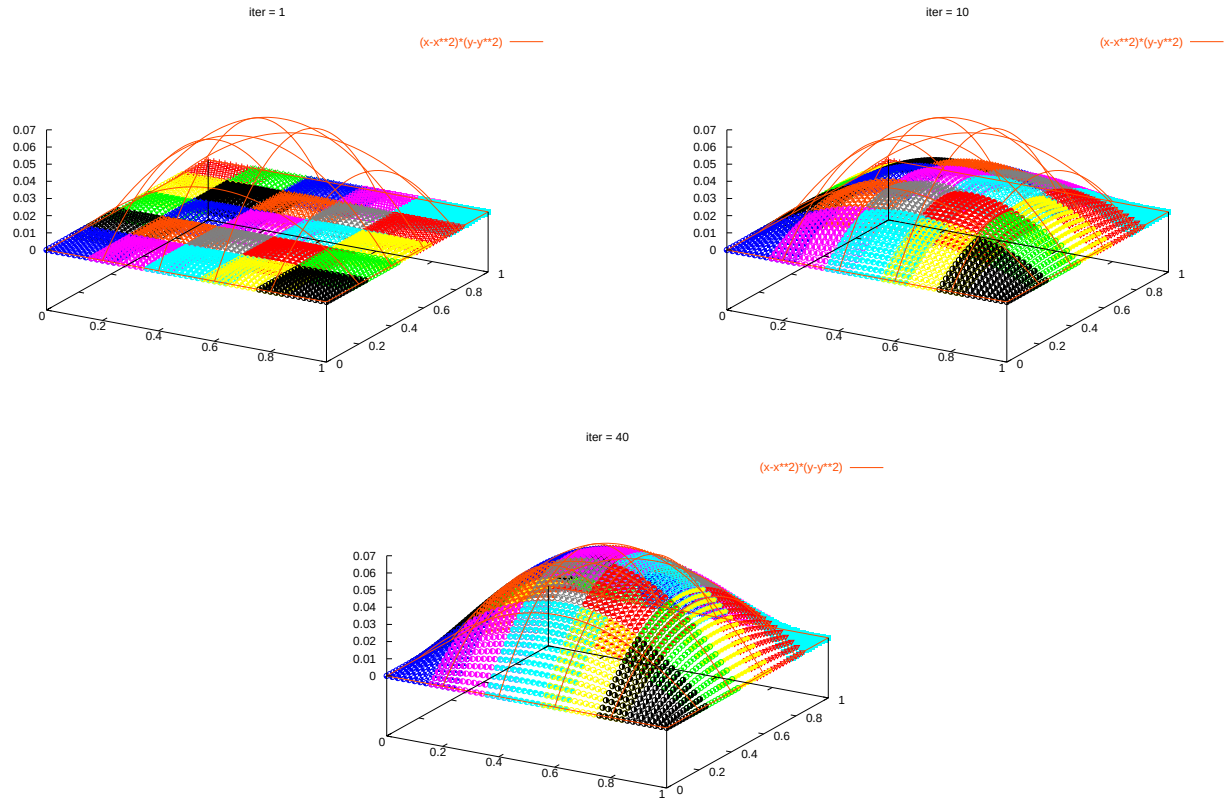

FIG. 4 – *Un calcul avec 25 sous-domaines. Solution initiale (en haut à gauche), solution calculée après 10 itérations (à droite) et solution finale après 40 itérations (en bas). La solution analytique est également représentée en rouge.*

Les premiers tests ont portés sur le modèle simplifié de la section 3. Le carré unité $\Omega = [0,1] \times [0,1]$, discrétisé avec un pas de 0,02 dans chaque direction, est subdivisé en 5×5 sous-domaines avec recouvrement de 2 mailles. L'évolution de la solution au cours des itérations de point fixe est présentée dans la figure 4.

Sur la figure 5, nous pouvons voir que l'on obtient un facteur d'accélération de l'ordre de 6 sur 10 processeurs. C'est un résultat très honnête sachant que l'implémentation actuelle n'est pas optimisée.

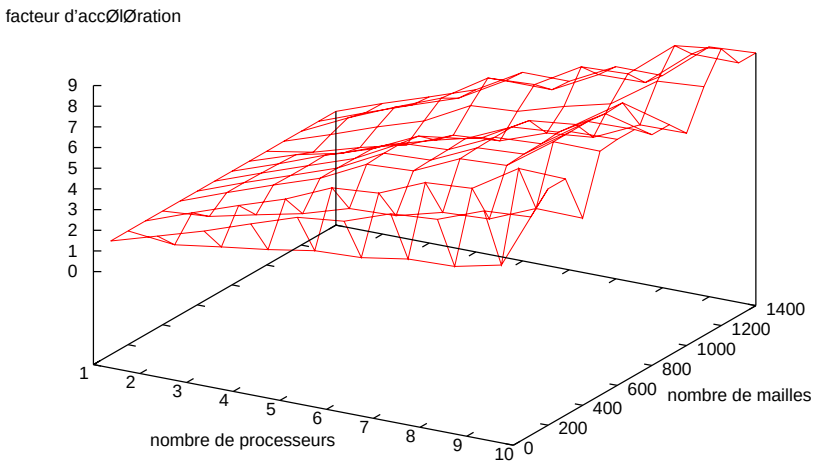

FIG. 5 – *Facteurs d'accélération sur 10 processeurs avec grilles de taille croissante : de 100×100 à 1400×1400.*

De plus, ces facteurs d'accélération semblent relativement insensibles à l'augmentation de la taille de la grille de calcul. Cela signifie que les conditions de l'expérience ne permettent pas de détecter de goulet d'étranglement dus aux communications (phase de projection réalisée par un unique processus) ; cependant, nous pensons qu'il faudra par la suite que les sous-domaines échangent directement les informations avec leurs seuls voisins.

6 Conclusions et perspectives

La réalisation d'une première maquette sur un exemple simplifié montre que le système OCamlP3l est parfaitement adapté à la parallélisation d'un code de calcul numériques. Les techniques mises en œuvre étant similaires, le couplage de codes entre également dans ses compétences. En effet, la mise à disposition de squelettes de base permet une écriture relativement compacte et plutôt claire de l'interface de couplage. Sans jamais se préoccuper des détails des communications entre processus, l'interface est d'abord mise au point en mode séquentiel et le passage au mode parallèle est ensuite "garanti". De plus, la possibilité d'écrire directement en OCaml les modules permettant de définir et manipuler les structures de données adaptées au problème assure un développement rapide et sûr de l'interface.

Le principal défaut est la quasi-absence de bibliothèques OCaml dédiées au calcul numérique (il faut tout écrire). La panoplie de squelettes OCamlP3l pourrait également être étoffée.

La poursuite de ces travaux va s'orienter selon trois axes : le généralisation de la maquette de couplage, l'optimisation de l'implémentation et la réalisation d'utilitaires.

Nous envisageons d'abord de généraliser la maquette. D'une part, en l'adaptant au couplage du programme LifeV décrit précédemment (Laplacien 3D sur un maillage non structuré) :

- choix des types OCaml et implémentation du module OCaml permettant la gestion des différentes structures de données nécessaires (maillages, valeurs aux interfaces à échanger),

- intégration de la projection L^2 dans le cas où les maillages ne se raccordent pas,
- implémentation complète et innovante de l'algorithme Bi-CGStab en OCamlP3l,
- appel à un logiciel graphique adapté aux maillages 3D non structurés, par exemple Medit développé au Projet Gamma.

D'autre part, nous devons enrichir l'offre algorithmique pour gagner en crédibilité :

- implémentation d'autres techniques de couplage, par exemple la méthode Mortar (qui revient à modifier l'opérateur de projection),
- implémentation d'autres algorithmes d'inversion de système linéaires, par exemple GMRES.

Il nous faudra également engager une réflexion sur une généralisation de la maquette de couplage à d'autres types de codes.

Mais nous projetons également d'optimiser l'implémentation. Tout d'abord en minimisant les goulets d'étranglement. Les tâches purement séquentielles doivent être identifiées, les autres doivent s'exécuter en parallèle :

- les communications sont actuellement réalisées par un unique processus qui doit donc collecter toute l'information avant de la redistribuer, mais elles devraient pouvoir s'effectuer de façon répartie ; cela pourrait conduire à la conception d'un nouveau squelette OCamlP3l,
- dans les algorithmes itératifs d'inversion de systèmes linéaires tel Bi-CGStab, seuls les produits scalaires sont des points de synchronisation et l'on doit assurer que toutes les autres tâches sont exécutées en parallèle.

Mais aussi en réfléchissant à des spécifications d'extensions d'OCamlP3l permettant, par exemple, de passer un squelette en argument d'une fonction OCaml. Cela rendrait plus souple l'intégration d'autres méthodes d'inversion de systèmes linéaires. Ainsi qu'à l'amélioration l'efficacité du code OCaml en travaillant sur les fonctions de plus bas niveau qui font le travail.

Enfin, nous pensons utile d'écrire un module OCaml de génération de maillages adaptés à la décomposition de domaines (en faisant appel à un mailleur existant), puis l'intégrer au coupleur.

Références

[1] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computations.* Research Monographs in Parallel and Distributed Computing. Pitman, 1989.

[2] M. Danelutto, R. Di Cosmo, X. Leroy, and S. Pelagatti. Ocamlp3l: (release 1.0). http://www.ocamlp3l.org/, 1998.

[3] M. Mancip. *Couplage de méthodes numériques pour les lois de conservation. Application au cas de l'injection.* PhD thesis, Université Paul Sabatier, Toulouse, 2001.

[4] V. Martin and S. Wagner. Simulation du stockage souterrain par des méthodes de décomposition de domaine non conformes. CEMRACS : projet ANDRA, 2001.

A Module Rdec

A.1 Interface

```
(* ********************************************************** *)
(* Module for the representation of 2D regular rectangular grids *)
(* with regular decomposition capabilities                    *)
(* Overlapping of the decomposition is parameterized          *)
(* ********************************************************** *)

(* ********************************************************************** *)
(* A config file contains a regular_decomposition record, ie:             *)
(*   a computational_rectangular_grid record:                             *)
(*     a rectangular_grid record:                                         *)
(*       a rectangle record:                                              *)
(*         x0 and y0, coordinates of the lower-left corner of the rectangle *)
(*         lx and ly, length of the rectangle in both directions          *)
(*       nx and ny, number of intervals in both directions                *)
(*     px and py, half the overlapping of the subdomains in both directions *)
(*   nsbdx and nsbdy, number of subdomains in both directions             *)
(*                                                                        *)
(* A valid regular_decomposition must satisfy:                            *)
(*   lx > 0. and ly > 0.                                                  *)
(*   nx > 0 and ny > 0                                                    *)
(*   px > 0 and py > 0                                                    *)
(*   px <= nx / nsbdx and py <= ny / nsbdy                                *)
(*   nsbdx > 0 and nsbdy > 0                                              *)
(*   nsbdx <= nx and nsbdy <= ny                                          *)
(*   nx mod nsbdx = 0 and ny mod nsbdy = 0                                *)
(* ********************************************************************** *)

type rectangle and rectangular_grid
and computational_rectangular_grid and regular_decomposition

val rectangle_of_rectangular_grid : rectangular_grid -> rectangle
val rectangle_of_computational_rectangular_grid :
    computational_rectangular_grid -> rectangle
val rectangle_of_regular_decomposition : regular_decomposition -> rectangle

val of_rectangle : rectangle -> float * float * float * float
val number_of_mesh_of_rectangular_grid : rectangular_grid -> int * int
val rectangular_grid_of_computational_rectangular_grid :
    computational_rectangular_grid -> rectangular_grid
val overlapping_of_computational_rectangular_grid :
    computational_rectangular_grid -> int * int
val computational_rectangular_grid_of_regular_decomposition :
    regular_decomposition -> computational_rectangular_grid
val number_of_subdomains_of_regular_decomposition :
    regular_decomposition -> int * int

val print_computational_rectangular_grid :
```

```
    out_channel -> computational_rectangular_grid -> unit

val input_config_file : string -> regular_decomposition
exception Wrong_Input of string
val valid_regular_decomposition : regular_decomposition -> unit

exception Out_of_range of int
val two_indices_numbering : int -> int -> int -> int * int

type subdomain = regular_decomposition * int
val rectangular_grid_of_subdomain :
    subdomain -> rectangular_grid
val computational_rectangular_grid_of_subdomain :
    subdomain -> computational_rectangular_grid

type connectivity_table
val connectivity_table_of_regular_decomposition :
    regular_decomposition -> connectivity_table
val east_neighbor : connectivity_table -> int -> int
val north_neighbor : connectivity_table -> int -> int
val south_neighbor : connectivity_table -> int -> int
val west_neighbor : connectivity_table -> int -> int

type boundary
val print_boundary : out_channel -> boundary -> unit
val read_boundary :
    (*Scanf.Scanning.scanbuf*) in_channel-> int * int -> boundary
val size_of_boundary : boundary -> int * int

type interface = boundary array
val make_zero_interface : regular_decomposition -> interface
val projection : connectivity_table -> interface -> interface

val for_all_subdomains : int * int -> (int -> 'a) -> unit
```

A.2 Implémentation

```
(* See "rdec.mli" *)

(* When using OCaml-3.06, formatted inputs with Scanf.fscanf need a format *)
(* argument with no ending newline character "\n", even if it is present   *)
(* in the read file; and Scanf.bscanf does not seem to work.               *)
(* With future releases of OCaml, one might have to add the "\n"...        *)

type rectangle = {x0 : float; y0 : float; lx : float; ly : float};;
type rectangular_grid = {domain : rectangle; nx : int; ny :int};;
type computational_rectangular_grid =
    {mesh : rectangular_grid; px : int; py : int};;
type regular_decomposition =
  {comp_grid : computational_rectangular_grid; nsbdx : int; nsbdy : int};;
```

```
type subdomain = regular_decomposition * int;;

let rectangle_of x0 y0 lx ly = {x0 = x0; y0 = y0; lx = lx; ly = ly};;

let rectangular_grid_of x0 y0 lx ly nx ny =
  let r = rectangle_of x0 y0 lx ly in
  {domain = r; nx = nx; ny = ny};;

let computational_rectangular_grid_of x0 y0 lx ly nx ny px py =
  let g = rectangular_grid_of x0 y0 lx ly nx ny in
  {mesh = g; px = px; py = py};;

let regular_decomposition_of x0 y0 lx ly nx ny px py nsbdx nsbdy =
  let g = computational_rectangular_grid_of x0 y0 lx ly nx ny px py in
  {comp_grid = g; nsbdx = nsbdx; nsbdy = nsbdy};;

let rectangle_of_rectangular_grid g =
  let {domain = {x0 = x0; y0 = y0; lx = lx; ly = ly}} = g in
  rectangle_of x0 y0 lx ly;;

let rectangle_of_computational_rectangular_grid {mesh = mesh} =
  rectangle_of_rectangular_grid mesh;;

let rectangle_of_regular_decomposition {comp_grid = comp_grid} =
  rectangle_of_computational_rectangular_grid comp_grid;;

let of_rectangle r = r.x0, r.y0, r.lx, r.ly;;
let number_of_mesh_of_rectangular_grid g = g.nx, g.ny;;
let rectangular_grid_of_computational_rectangular_grid g = g.mesh;;
let overlapping_of_computational_rectangular_grid g = g.px, g.py;;
let computational_rectangular_grid_of_regular_decomposition dec =
  dec.comp_grid;;
let number_of_subdomains_of_regular_decomposition dec = dec.nsbdx, dec.nsbdy;;

let print_rectangle oc r =
  Printf.fprintf oc "%f %f\n%f %f\n" r.x0 r.y0 r.lx r.ly;;

let print_rectangular_grid oc g =
  Printf.fprintf oc "%a%d %d\n" print_rectangle g.domain g.nx g.ny;;

let print_computational_rectangular_grid oc g =
  Printf.fprintf oc "%a%d %d\n" print_rectangular_grid g.mesh g.px g.py;;

let print_regular_decomposition oc dec =
  Printf.fprintf oc "%a%d %d\n"
    print_computational_rectangular_grid dec.comp_grid dec.nsbdx dec.nsbdy;;

let read_rectangle ib = Scanf.(*b*)fscanf ib "%f %f\n%f %f" rectangle_of;;

let read_rectangular_grid ib =
```

```
  let r = read_rectangle ib in
  Scanf.(*b*)fscanf ib "%d %d" (fun nx ny -> {domain = r; nx = nx; ny = ny});;

let read_computational_rectangular_grid ib =
  let g = read_rectangular_grid ib in
  Scanf.(*b*)fscanf ib "%d %d" (fun px py -> {mesh = g; px = px; py = py});;

let read_regular_decomposition ib =
  let g = read_computational_rectangular_grid ib in
  Scanf.(*b*)fscanf ib "%d %d"
    (fun nsbdx nsbdy -> {comp_grid = g; nsbdx = nsbdx; nsbdy = nsbdy});;

let input_config_file fname =
  let ic = open_in fname in
  let ib = (*Scanf.Scanning.from_channel*) ic in
  let dec = read_regular_decomposition ib in
  close_in ic;
  dec;;

exception Wrong_Input of string;;

let valid_regular_decomposition
    {comp_grid =
     {mesh = {domain = {lx = lx; ly = ly}; nx = nx; ny = ny};
      px = px; py = py};
     nsbdx = nsbdx; nsbdy = nsbdy} =
  if lx <= 0. || ly <= 0. then
    raise (Wrong_Input "Negative length!");
  if nx <= 0 || ny <= 0 then
    raise (Wrong_Input "Negative number of intervals!");
  if px <= 0 || py <= 0 then
    raise (Wrong_Input "Negative overlapping!");
  if px > nx / nsbdx || py > ny / nsbdy then
    raise (Wrong_Input "Overlapping is too large!");
  if nsbdx <= 0 || nsbdy <= 0 then
    raise (Wrong_Input "Negative number of subdomains!");
  if nsbdx > nx || nsbdy > ny then
    raise (Wrong_Input "Too many subdomains!");
  if nx mod nsbdx <> 0 || ny mod nsbdy <> 0 then
    raise (Wrong_Input
             "Number of subdomains do not divide number of intervals!");;

type location = EastSide | NorthSide | SouthSide | WestSide;;
module OrderedLocation = struct type t = location let compare = compare end;;
module Position = Set.Make (OrderedLocation);;
type position = Position.t;;

exception Out_of_range of int;;

let two_indices_numbering nx ny i =
```

```
    if i < 0 || i > nx * ny - 1 then raise (Out_of_range i) else
    (i mod nx, i / nx);;

let rectangular_grid_of_subdomain (dec, i) =
  let {comp_grid =
       {mesh = {domain = {x0 = x0; y0 = y0; lx = lx; ly = ly};
                nx = nx; ny = ny}};
       nsbdx = nsbdx; nsbdy = nsbdy} = dec in
  let ix, iy = two_indices_numbering nsbdx nsbdy i in
  let lxi = lx /. float_of_int nsbdx and lyi = ly /. float_of_int nsbdy in
  rectangular_grid_of
    (x0 +. float_of_int ix *. lxi) (y0 +. float_of_int iy *. lyi)
    lxi lyi (nx / nsbdx) (ny / nsbdy);;

let position_of_subdomain ({nsbdx = nsbdx; nsbdy = nsbdy}, i) =
  let ix, iy = two_indices_numbering nsbdx nsbdy i in
  let locations_to_add = [(ix = 0, WestSide); (ix = nsbdx - 1, EastSide);
                          (iy = 0, SouthSide); (iy = nsbdy - 1, NorthSide)] in
  let rec loop accu = function
    | [] -> accu
    | (b, e) :: l -> if b then loop (Position.add e accu) l else loop accu l
  in
  loop Position.empty locations_to_add;;

let complement pos =
  let fullset = Position.add WestSide (Position.add SouthSide
     (Position.add NorthSide (Position.add EastSide Position.empty))) in
  Position.diff fullset pos;;

let east_extension_of_computational_rectangular_grid
    {mesh = {domain = {x0 = x0; y0 = y0; lx = lx; ly = ly}; nx = nx; ny = ny};
     px = px; py = py} =
  let dx = lx /. float_of_int nx in
  computational_rectangular_grid_of
    x0 y0 (lx +. float_of_int px *. dx) ly (nx + px) ny px py;;

let north_extension_of_computational_rectangular_grid
    {mesh = {domain = {x0 = x0; y0 = y0; lx = lx; ly = ly}; nx = nx; ny = ny};
     px = px; py = py} =
  let dy = ly /. float_of_int ny in
  computational_rectangular_grid_of
    x0 y0 lx (ly +. float_of_int py *. dy) nx (ny + py) px py;;

let south_extension_of_computational_rectangular_grid
    {mesh = {domain = {x0 = x0; y0 = y0; lx = lx; ly = ly}; nx = nx; ny = ny};
     px = px; py = py} =
  let dy = ly /. float_of_int ny in
  computational_rectangular_grid_of
    x0 (y0 -. float_of_int py *. dy) lx (ly +. float_of_int py *. dy)
    nx (ny + py) px py;;
```

```
let west_extension_of_computational_rectangular_grid
    {mesh = {domain = {x0 = x0; y0 = y0; lx = lx; ly = ly}; nx = nx; ny = ny};
     px = px; py = py} =
  let dx = lx /. float_of_int nx in
   computational_rectangular_grid_of
    (x0 -. float_of_int px *. dx) y0 (lx +. float_of_int px *. dx) ly
    (nx + px) ny px py;;

let computational_rectangular_grid_of_subdomain subd =
  let pos = position_of_subdomain subd in
  let dec = fst subd in
  let g0 = {mesh = rectangular_grid_of_subdomain subd;
            px = dec.comp_grid.px; py = dec.comp_grid.py} in
  let ext_of_grid g = function
    | EastSide -> east_extension_of_computational_rectangular_grid g
    | NorthSide -> north_extension_of_computational_rectangular_grid g
    | SouthSide -> south_extension_of_computational_rectangular_grid g
    | WestSide -> west_extension_of_computational_rectangular_grid g
  in
  let rec loop accu = function
    | [] -> accu
    | cp :: l -> loop (ext_of_grid accu cp) l
  in
  loop g0 (Position.elements (complement pos));;

type neighbor = int;;
type neighborhood = {east_n : neighbor; north_n : neighbor;
                     south_n : neighbor; west_n : neighbor};;
type connectivity_table = neighborhood array;;

let neighborhood_of e n s w =
  {east_n = e; north_n = n; south_n = s; west_n = w};;

let change_east_neighbor
    {east_n = e; north_n = n; south_n = s; west_n = w} i =
  neighborhood_of i n s w;;

let change_north_neighbor
    {east_n = e; north_n = n; south_n = s; west_n = w} i =
  neighborhood_of e i s w;;

let change_south_neighbor
    {east_n = e; north_n = n; south_n = s; west_n = w} i =
  neighborhood_of e n i w;;

let change_west_neighbor
    {east_n = e; north_n = n; south_n = s; west_n = w} i =
  neighborhood_of e n s i;;
```

```
let neighborhood_of_subdomain (dec, i) =
  let pos = position_of_subdomain (dec, i) in
  let n = dec.nsbdx in
  let nbh0 = {east_n = - 1; north_n = - 1; south_n = - 1; west_n = - 1} in
  let change_neighborhood nbh = function
    | EastSide -> change_east_neighbor nbh (i + 1)
    | NorthSide -> change_north_neighbor nbh (i + n)
    | SouthSide -> change_south_neighbor nbh (i - n)
    | WestSide -> change_west_neighbor nbh (i - 1)
  in
  let rec loop accu = function
    | [] -> accu
    | cp :: l -> loop (change_neighborhood accu cp) l
  in
  loop nbh0 (Position.elements (complement pos));;

let connectivity_table_of_regular_decomposition dec =
  let n = dec.nsbdx * dec.nsbdy in
  Array.init n (fun i -> neighborhood_of_subdomain (dec, i));;

let east_neighbor tbl i = let {east_n = e} = tbl.(i) in e;;
let north_neighbor tbl i = let {north_n = n} = tbl.(i) in n;;
let south_neighbor tbl i = let {south_n = s} = tbl.(i) in s;;
let west_neighbor tbl i = let {west_n = w} = tbl.(i) in w;;

type edge_values = float array;;
type boundary = {east_ev : edge_values; north_ev : edge_values;
                 south_ev : edge_values; west_ev : edge_values};;
type interface = boundary array;;

let print_edge_values oc ev =
  Array.iter (fun v -> Printf.fprintf oc "%f\n" v) ev;;

let print_boundary oc bd =
  Printf.fprintf oc "%a%a%a%a"
    print_edge_values bd.east_ev print_edge_values bd.north_ev
    print_edge_values bd.south_ev print_edge_values bd.west_ev;;

let read_edge_values ib sz =
  Array.init sz (fun i -> Scanf.(*b*)fscanf ib "%f" (fun v -> v));;

let read_boundary ib (nx, ny) =
  let e = read_edge_values ib ny in
  let n = read_edge_values ib nx in
  let s = read_edge_values ib nx in
  let w = read_edge_values ib ny in
  {east_ev = e; north_ev = n; south_ev = s; west_ev = w};;

let size_of_boundary bd = Array.length bd.north_ev, Array.length bd.east_ev;;
```

```
let make_zero_edge_values n = Array.make n 0.;;

let make_zero_boundary nx ny =
  let e = make_zero_edge_values ny in
  let n = make_zero_edge_values nx in
  let s = make_zero_edge_values nx in
  let w = make_zero_edge_values ny in
  {east_ev = e; north_ev = n; south_ev = s; west_ev = w};;

let make_zero_interface dec =
  let f i =
    let {mesh = {nx = nx; ny = ny}} =
      computational_rectangular_grid_of_subdomain (dec, i) in
    make_zero_boundary (nx + 1) (ny + 1) in
  Array.init (dec.nsbdx * dec.nsbdy) f;;

let projection tbl intf =
  let f i =
    let je = east_neighbor tbl i and jn = north_neighbor tbl i in
    let js = south_neighbor tbl i and jw = west_neighbor tbl i in
    let e =
      if je <> - 1 then Array.copy intf.(je).west_ev else
      Array.make (Array.length intf.(i).east_ev) 0.
    and n =
      if jn <> - 1 then Array.copy intf.(jn).south_ev else
      Array.make (Array.length intf.(i).north_ev) 0.
    and s =
      if js <> - 1 then Array.copy intf.(js).north_ev else
      Array.make (Array.length intf.(i).south_ev) 0.
    and w =
      if jw <> - 1 then Array.copy intf.(jw).east_ev else
      Array.make (Array.length intf.(i).west_ev) 0. in
    {east_ev = Array.copy e; north_ev = Array.copy n;
     south_ev = Array.copy s; west_ev = Array.copy w} in
  Array.init (Array.length tbl) f;;

let for_all_subdomains (nsbdx, nsbdy) to_do =
  let num = ref (nsbdx - 1 + nsbdx * nsbdy) in
  for j = nsbdy - 1 downto 0 do
    num := !num - 2 * nsbdx;
    for i = 0 to nsbdx - 1 do
      num := !num + 1;
      to_do !num;
    done;
  done;;
```

B Maquette de couplage

```
(* Program solving Poisson's equation on a rectangle with homogeneous
   Dirichlet boundary conditions, ie
            { - Laplacian u = f  on Omega,
            {             u = 0  on Gamma = boundary of Omega.
   The Partial Derivative Equation (PDE) is solved using Domain Decomposition
   with overlapping: firstly, on each subdomain, we input some specific
   Dirichlet conditions, solve the PDE and then output Dirichlet conditions
   for the neighbor subdomains; secondly, the solution is searched for as the
   fix point of the previously defined Dirichlet to Dirichlet operator.
   Computation on each subdomain can be run in parallel.

   The OCamlP3l implementation uses the loop skeleton for the fix point
   and the mapvector skeleton for the concurrent computation on each subdomain.
   The poisson's solver is a C program implementing an implicit 5-point scheme
   which is spawned and connected via stdin/stdout to
   computation_on_a_subdomain in a seq skeleton.
   Graphical capabilities using gnuplot are also spawned.

   The Rdec module provides functions to manipulate regular decompositions
   of regular rectangular grids.

   IMPORTANT:
   - remember to fflush data output in the C program, or you will block!
   - remember not to rely on global references/arrays.
 *)

(* To compile, run the following commands:
   gcc -o poisson poisson.c -lm            # for the Poisson's solver
   ocamlc -c rdec.mli; ocamlc -c rdec.ml   # for the Rdec module
   ocamlp3lcc -seq coupling.ml rdec.cmo    # for sequential run
   ocamlp3lcc -par coupling.ml rdec.cmo    # for parallel run
 *)

(* Execution is controlled by data read in a config file     *)
(* This file should contain:                                 *)
(* - the name of the file defining the regular decomposition *)
(* - the basename for result files                           *)
(* - the number of iterations for the fix point              *)
(* - the refreshment step for the graphical device           *)
(* - the number of seconds to sleep before terminating       *)
let ic = open_in "config";;
let ib = (*Scanf.Scanning.from_channel*) ic;;
let config, resu, itermax, plot_step, a_while =
  Scanf.(*b*)fscanf ib "%s\n%s\n%d\n%d\n%d"
    (fun fn fbn imax step t -> fn, fbn, imax, step, t);;
close_in ic;;

(* See "rdec.mli" for the content of the config file *)
```

```
let dec = Rdec.input_config_file config;;
Rdec.valid_regular_decomposition dec;;
let table = Rdec.connectivity_table_of_regular_decomposition dec;;
let intf0 = Rdec.make_zero_interface dec;;
let nsbdx, nsbdy = Rdec.number_of_subdomains_of_regular_decomposition dec;;
let nbproc = nsbdx * nsbdy;;

let generate_input_stream =
  let k = ref 0 in
  (fun () ->
    k := !k + 1;
    if !k = 1 then Array.mapi (fun i bd -> i, bd) intf0 else
    raise End_of_file);;

let print_result v =
  let f = (fun (num, _) ->
    Printf.eprintf "Computation on subdomain %d has completed\n" num) in
  Array.iter f v; flush stderr;;

let sleep n = (fun () -> Unix.sleep n);;

let convergence =
  let iter = ref 0 in
  let t0 = ref (Unix.gettimeofday ()) in
  (fun _ ->
    iter := !iter + 1;
    let dt = Unix.gettimeofday () -. !t0 in
    Printf.eprintf "iter = %d (t = %f)\n" !iter dt; flush stderr;
    !iter < itermax);;

let spawn_it command bin cout =
  match !bin, !cout with
    | Some ib, Some oc -> ib, oc
    | _ ->
       let ic, oc = Unix.open_process command in
       let ib = (*Scanf.Scanning.from_channel*) ic in
       bin := Some ib; cout := Some oc;
       ib, oc;;

let computation_on_a_subdomain =
  let bin = ref None and cout = ref None in
  (fun (num, bd) ->
    let ib, oc = spawn_it "poisson" bin cout in
    let g = Rdec.computational_rectangular_grid_of_subdomain (dec, num) in
    Rdec.print_computational_rectangular_grid oc g;
    Printf.fprintf oc "%s\n" (resu ^ string_of_int num);
    Rdec.print_boundary oc bd;
    flush oc;
    let bd = Rdec.read_boundary ib (Rdec.size_of_boundary bd) in
    (num, bd));;
```

```
let proj v =
  Array.mapi (fun i d -> i, d) (Rdec.projection table (Array.map snd v));;

let make_gnuplot_command iter nsbdx nsbdy =
  let siter = "iter = " ^ string_of_int iter in
  let snxp1 = string_of_int (nsbdx + 1) in
  let snyp1 = string_of_int (nsbdy + 1) in
  let gnuplot = ref
      ("set title '" ^ siter ^ "'\n" ^
       "set isosample " ^ snxp1 ^ ", " ^ snyp1 ^ "\n" ^
       "splot [0:1][0:1][0:0.07]") in
  let to_do n =
    let sn = string_of_int n in
    let resun = resu ^ sn and resubn = resu ^ "B" ^ sn in
    if Sys.file_exists resun then Sys.rename resun resubn else
    begin
      Printf.eprintf "%s not found!\n" resun; flush stderr;
    end;
    gnuplot := !gnuplot ^ " '" ^ resubn ^ "' notitle w p," in
  Rdec.for_all_subdomains (nsbdx, nsbdy) to_do;
  gnuplot := !gnuplot ^ " (x-x**2)*(y-y**2) w l\n";
  !gnuplot;;

let plot =
  let bin = ref None and cout = ref None in
  let iter = ref 0 in
  (fun v ->
    let ic, oc = spawn_it "gnuplot" bin cout in
    iter := !iter + 1;
    if !iter mod plot_step = 0 then
      begin
        let gpl = make_gnuplot_command !iter nsbdx nsbdy in
        Printf.fprintf oc "%s\n" gpl; flush oc;
      end;
    v);;

let program () =
  startstop
    (generate_input_stream, ignore)
    (print_result, ignore, sleep a_while)
    (loop (convergence, mapvector (seq computation_on_a_subdomain, nbproc)
         ||| seq proj ||| seq plot))
in
pardo program;;
```

C Code de résolution de l'équation de Poisson

```
/* Fast Poisson solver using SOR on a domain Omega
     Laplacian(u) = f on Omega
                u  = g on Gamma */

#include <stdio.h>
#include <math.h>
#include <malloc.h>

#define MAX(a,b) ((a)>=(b) ? (a) : (b))
#define PI 3.1415926535589793

float maxnum(int m,float *x) {
  int k;
  float z;
  z = MAX(fabs(x[1]),fabs(x[2]));
  for (k=3; k<=m; k++)
    z = MAX(fabs(x[k]),z);
  return z; }

float f(float x,float y) {
  return 2*(x*x - x + y*y - y); }

float *vecalloc(int low,int high) {
  float *x;
  x = (float *)calloc((unsigned)(high-low)+1,sizeof(float));
  if (x==NULL) {
    fprintf(stderr,"unable to allocate memory");
    exit(1); }
  return (x-low); }

float **mataIloc(int rowlow,int rowhigh,int collow,int colhigh) {
  int k;
  float **x;
  x = (float **)calloc((unsigned)(rowhigh-rowlow+1),sizeof(float *));
  if (x==NULL) {
    fprintf(stderr,"unable to allocate memory");
    exit(1); }
  x -= rowlow;
  for(k=rowlow; k<=rowhigh; k++) {
    x[k] = (float *)calloc((unsigned)(colhigh-collow+1),sizeof(float));
    if (x[k]==NULL) {
      fprintf(stderr,"unable to allocate memory");
      exit(1); }
    x[k] -= collow; }
  return x; }

main() {
```

```
int   i,j,nx,ny,px,py,iter,itermax;
float x0,y0,lx,ly,dx,dy,eta,w,alpha,beta,rsum;
float *e,*x,*y,**uold,**unew,**ff;
float f(float x,float y);
float maxnum(int m,float *x);
char  resultat[255];
FILE  *fp;

while (1) {

  /*****************************************/
  /* LECTURE DE LA GRILLE DE CALCUL SUR STDIN */
  /*****************************************/

  scanf("%f",&x0); /* abscisse du coin inferieur gauche */
  scanf("%f",&y0); /* ordonnee du coin inferieur gauche */
  scanf("%f",&lx); /* extension horizontale du rectangle */
  scanf("%f",&ly); /* extension verticale du rectangle */
  scanf("%d",&nx); /* nombre de mailles horizontalles */
  scanf("%d",&ny); /* nombre de mailles verticales */
  scanf("%d",&px); /* recouvrement horizontal pour les communication */
  scanf("%d",&py); /* recouvrement vertical pour les communication */
  scanf("%256s",resultat); /* nom du fichier de resultats */

  /* */

  dx = lx/nx; dy = ly/ny;
  eta = 1. - 2.*sin(PI/(2.*nx))*sin(PI/(2.*ny));
  w = 2./(1. + sqrt(1. - eta*eta));
  alpha = 0.25*w; beta = 1. - w;

  e = vecalloc(1,nx-1);
  uold = matalloc(0,nx,0,ny);
  unew = matalloc(0,nx,0,ny);
  x = vecalloc(0,nx); y = vecalloc(0,ny);
  ff = matalloc(1,nx-1,1,ny-1);

  for (i=0; i<=nx; i++) {x[i] = x0 + i*dx; }
  for (j=0; j<=ny; j++) {y[j] = y0 + j*dy; }
  for (i=1; i<=nx-1; i++) {
    for (j=1; j<=ny-1; j++) {
      ff[i][j] = dx*dy*f(x[i],y[j]); } }

  /*************************************************/
  /* LECTURE DE LA CONDITION DE DIRICHLET SUR STDIN */
  /* Attention, chaque coin est lu deux fois !      */
  /*************************************************/

  for (j=0; j<=ny; j++) { /* bord "est" */
    scanf("%f",&uold[nx][j]); unew[nx][j] = uold[nx][j]; }
```

```
for (i=0; i<=nx; i++) { /* bord "nord" */
  scanf("%f",&uold[i][ny]); unew[i][ny] = uold[i][ny]; }
for (i=0; i<=nx; i++) { /* bord "sud" */
  scanf("%f",&uold[i][0]); unew[i][0] = uold[i][0]; }
for (j=0; j<=ny; j++) { /* bord "ouest" */
  scanf("%f",&uold[0][j]); unew[0][j] = uold[0][j]; }

/****************************************/
/* RESOLUTION {Laplacien(u) = f sur Omega */
/*            {          u  = g sur Gamma */
/****************************************/

iter = 0; itermax = 50;

do {
  for (i=1; i<=nx-1; i++) {
    for (j=1; j<=ny-1; j++) {
      unew[i][j] = alpha*(uold[i+1][j] + uold[i][j+1] \
                          + unew[i-1][j] + unew[i][j-1] \
                          - ff[i][j])
        + beta*uold[i][j]; } }

  /*for (i=1; i<=nx-1; i++) {
    rsum=0.0;
    for(j=1; j<=ny-1; j++) {
      rsum += fabs(unew[i][j] - uold[i][j]);
      } e[i] = rsum; }*/

  for (i=1; i<=nx-1; i++) {
    for(j=1; j<=ny-1; j++) {
      uold[i][j] = unew[i][j]; } }

  iter++;

  /*} while (maxnum(nx,e)>5.0e-9);*/
} while (iter<itermax);

/* Complexite = 7*(nx-1)*(ny-1)*itermax operations flottantes */

/****************************************/
/* ECRITURE DE LA SOLUTION DANS UN FICHIER */
/****************************************/

fp = fopen(resultat,"w");
for (i=0; i<=nx; i++) {
  for (j=0; j<=ny; j++) {
    fprintf(fp,"%f %f %f\n",x[i],y[j],uold[i][j]); } }
fclose(fp);

/****************************************/
```

```
    /* ECRITURE DES COMMUNICATIONS SUR STDOUT */
    /******************************************/

    for (j=0; j<=ny; j++) { /* cote "est" */
      printf("%f\n",uold[nx-2*px][j]); }
    for (i=0; i<=nx; i++) { /* cote "nord" */
      printf("%f\n",uold[i][ny-2*py]); }
    for (i=0; i<=nx; i++) { /* cote "sud" */
      printf("%f\n",uold[i][2*py]); }
    for (j=0; j<=ny; j++) { /* cote "ouest" */
      printf("%f\n",uold[2*px][j]); }
    fflush(stdout);

  } /* end while */

  return 0;

} /* end main */
```

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399