Enseirb-matmeca

C.H.P - Projet de programmation

— Parallélisation d'algorithme numérique —

Thomas Arrivé
Antoine Delaire
Luc Védie

Encadrant :
Héloïse Beaugendre

Bordeaux INP
ENSEIRB
MATMECA

15 mai 2015 à Talence

tarrive@enseirb-matmeca.fr adelaire@enseirb-matmeca.fr lvedie@enseirb-matmeca.fr

Table des matières

Introduction

L'objectif est d'implémenter un programme (en Fortran 90) permettant de résoudre numériquement l'équation 2D selon des paramètres pré-établis, et de rendre ce programme exécutable en programmation parallèle en utilisant l'environnement MPI afin étudier le gain de performance réalisable de cette manière. Le projet se base sur l'équation de la chaleur décrite ci-dessous en première partie.

Le présent rapport est donc composé d'une analyse mathématique du problème numérique, puis d'une réflexion sur la parallélisation du programme, et enfin d'une validation des résultat obtenus et l'analyse des performances en temps.

I Analyse du problème

I.1 Écriture du schéma

On cherche à résoudre l'équation de la chaleur dans un domaine $\Omega = [0,L_x]\text{x}[0,L_y]$.
Soit le système suivant :

$$\left\{\begin{array}{rcll} \partial_t u(x,y,t) - D\Delta u(x,y,t) & = & f(x,y,t) & \\ u|_{\Gamma_0} & = & g(x,y,t) & \Gamma_0 = \{(x,y) \in \Omega \mid y = 0 \text{ ou } y = L_y\} \\ u|_{\Gamma_1} & = & h(x,y,t) & \Gamma_1 = \{(x,y) \in \Omega \mid x = 0 \text{ ou } x = L_x\} \end{array}\right. \quad (1)$$

Avec f, g et h fonctions données (c.f partie 2). En approximant le terme Δu par différences finies centrées, on obtient le développement suivant :

$$\left\{\begin{array}{l} \partial_{xx} u(x_i,y_j,t_n) = \dfrac{u(x_{i-1},y_j,t_n) - 2u(x_i,y_j,t_n) + u(x_{i+1},y_j,t_n)}{\Delta x^2} + O(\Delta x^2) \quad , \; x_i = i.\Delta x = i.\dfrac{L_x}{N_x+1} \\ \partial_{yy} u(x_i,y_j,t_n) = \dfrac{u(x_i,y_{j-1},t_n) - 2u(x_i,y_j,t_n) + u(x_i,y_{j+1},t_n)}{\Delta y^2} + O(\Delta y^2) \quad , \; y_j = j.\Delta y = j.\dfrac{L_y}{N_y+1} \end{array}\right.$$

Avec $t_n = n.dt$

Soit en réécrivant l'équation (1) :

$$\begin{array}{rl} \dfrac{\partial u(x_i,y_j,t_n)}{\partial t} = & D\left(\dfrac{u(x_{i-1},y_j,t_n) - 2u(x_i,y_j,t_n) + u(x_{i+1},y_j,t_n)}{\Delta x^2}\right. \\ & \left. + \dfrac{u(x_i,y_{j-1},t_n) - 2u(x_i,y_j,t_n) + u(x_i,y_{j+1},t_n)}{\Delta y^2}\right) \\ & + f(x_i,y_j,t_n) + O(\Delta x^2) + O(\Delta y^2) \end{array}$$

On rappelle la formulation du schéma d'Euler implicite pour une équation de la forme $y'(x,t) = f(y(x,t))$:
$y^{n+1} = y^n + \Delta t.f(y^{n+1})$ avec $y^n = y(x,t_n)$ et $t_{n+1} = t_n + dt$

En posant $u^n_{i,j} \simeq u(x_i,y_j,t_n)$ on obtient un schéma d'Euler implicite d'ordre deux en espace :

$$u^{n+1}_{i,j} = u^n_{i,j} + \Delta t.f^{n+1}_{i,j} + \Delta t.D\left(\frac{u^{n+1}_{i-1,j} - 2u^{n+1}_{i-1,j} + u^{n+1}_{i+1,j}}{\Delta x^2} + \frac{u^{n+1}_{i,j-1} - 2u^{n+1}_{i,j} + u^{n+1}_{i,j+1}}{\Delta y^2}\right)$$

Enfin, en récrivant le système :

$$\begin{array}{rl} \left(1 + 2.D\Delta t\left(\dfrac{1}{\Delta x^2} + \dfrac{1}{\Delta y^2}\right)\right) u^{n+1}_{i,j} - \dfrac{\Delta t}{\Delta x^2}.D(u^{n+1}_{i-1,j} + u^{n+1}_{i+1,j}) = & u^n_{i,j} + \Delta t.f^{n+1}_{i,j} \\ - \dfrac{\Delta t}{\Delta y^2}.D(u^{n+1}_{i,j-1} + u^{n+1}_{i,j+1}) & \end{array} \quad (2)$$

I.2 Formulation matricielle - Terme principal

Afin de parvenir à un système A.U = G, on pose $U^{n+1} = \begin{pmatrix} u_{1,1}^{n+1} \\ \vdots \\ u_{N_x,1}^{n+1} \\ u_{1,2}^{n+1} \\ \vdots \\ u_{i,j}^{n+1} \\ \vdots \\ u_{N_x,N_y}^{n+1} \end{pmatrix} = \begin{pmatrix} u_1^{n+1} \\ \vdots \\ u_{N_x}^{n+1} \\ u_{N_x+1}^{n+1} \\ \vdots \\ u_k^{n+1} \\ \vdots \\ u_{N_x.N_y}^{n+1} \end{pmatrix} \quad k = i + (j-1).N_x$

La partie gauche du système d'équation (2) en U^{n+1} peut alors se traduire par un produit $A.U^{n+1}$ avec A matrice tri-diagonale par bloc :
- Le bloc central est composé du terme principal diagonal α et des sur/sous diagonales β
- Le bloc sur/sous diagonal est lui-même diagonal de coefficient γ (le terme non nul se situe donc sur la N_x-ième sur/sous diagonale de la matrice)

$$\alpha = 1 + 2D.\Delta t \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \right), \qquad \beta = -D.\frac{\Delta t}{\Delta x^2}, \qquad \gamma = -D.\frac{\Delta t}{\Delta y^2} \tag{3}$$

Il convient désormais de traiter le problème des conditions limites, de type Dirichlet. Nous avons choisi de les introduire dans le second membre de l'équation. Ainsi, chaque coefficient faisant appel à une valeur extrémale connue (et par construction, absente du vecteur) sera annulée dans la matrice A.

En x :
$\beta = 0 \quad \forall (i,j) \in \{(kN_x - 1\ ,\ kN_x)\ ,\ k = 1..N_y - 1\}$ Annulation en x = 0
$\beta = 0 \quad \forall (i,j) \in \{(kN_x\ ,\ kN_x + 1)\ ,\ k = 1..N_y - 1\}$ Annulation en x = L_x
En y :

Contrairement au problème précédent, les indices concernés par cette condition limite sont strictement en dehors de la matrice. Les conditions limites sont donc déjà virtuellement traitées (dans la matrice, du moins).

Finalement on obtient A en posant les matrices carrées suivante :

$$B = \begin{pmatrix} \alpha & \beta & & & & \\ \beta & \alpha & \beta & & (0) & \\ & \beta & \alpha & \beta & & \\ & & \ddots & \ddots & \ddots & \\ & (0) & & \beta & \alpha & \beta \\ & & & & \beta & \alpha \end{pmatrix}_{N_x} \qquad C = \begin{pmatrix} \gamma & & & & \\ & \gamma & & (0) & \\ & & \ddots & & \\ & (0) & & \gamma & \\ & & & & \gamma \end{pmatrix}_{N_x}$$

$$A = \begin{pmatrix} B & C & & & & \\ C & B & C & & (0) & \\ & C & B & C & & \\ & & \ddots & \ddots & \ddots & \\ & (0) & & C & B & C \\ & & & & C & B \end{pmatrix}_{N_x \times N_y}$$

On remarque un élément important pour la suite du problème : la matrice A est symétrique à diagonale fortement dominante, elle est donc inversible d'après le théorème de Gerschgorin.

I.3 Second membre et conditions limites

De même que précédemment, on pose : $U^n = \begin{pmatrix} u_1^n \\ \vdots \\ u_k^n \\ \vdots \\ u_{N_x.N_y}^n \end{pmatrix}$ $F^{n+1} = \begin{pmatrix} f_1^{n+1} \\ \vdots \\ f_k^{n+1} \\ \vdots \\ f_{N_x.N_y}^{n+1} \end{pmatrix}$ $k = i + (j-1).N_x$

On peut alors écrire l'équation (2) sous la forme $A.U^{n+1} = U^n + F^{n+1}$, sous réserve de traiter correctement les conditions limites.

En y : $\forall i \in \{1..N_x\}$

$$\begin{cases} f(x_i, 0, t_{n+1}) = g(x_i, 0, t_{n+1}) & \Rightarrow F^{n+1}(i) = f_i^{n+1} + D\dfrac{\Delta t}{\Delta y^2}.g(x_i, 0, t_{n+1}) \\ f(x_i, L_y, t_{n+1}) = g(x_i, L_y, t_{n+1}) & \Rightarrow F^{n+1}((N_x-1)\times N_y + j) = f_{(N_x-1)\times N_y+j}^{n+1} + D\dfrac{\Delta t}{\Delta x^2}.h(L_x, y_j, t_{n+1}) \end{cases}$$

En x :

$$\begin{cases} f(0, y_j, t_{n+1}) = g(0, y_j, t_{n+1}) & \Rightarrow F^{n+1}(j\times N_x + 1) = f_{j\times N_x+1}^{n+1} + D\dfrac{\Delta t}{\Delta x^2}.g(0, y_j, t_{n+1}) \quad \forall j \in \{0..N_y - 1\} \\ f(L_x, y_j, t_{n+1}) = g(L_x, y_j, t_{n+1}) & \Rightarrow F^{n+1}(j\times N_x) = f_{j\times N_x}^{n+1} + D\dfrac{\Delta t}{\Delta x^2}.g(L_x, y_j, t_{n+1}) \quad \forall j \in \{1..N_y\} \end{cases}$$

A noter que ces deux termes induits par le Laplacien discrétisé apparaissent simultanément aux coins du domaine (en (0 ;0) , (0 ;L_y) , (L_x ;0) et (L_x,L_y))

La forme finale du second membre est alors :

$$F = \begin{pmatrix} f_1^n & + D\left(\dfrac{\Delta t}{\Delta x^2}.h(0, y_1, t_n) + \dfrac{\Delta t}{\Delta y^2}.g(x_1, 0, t_n)\right) \\ \vdots & \\ f_{N_x}^n & + D\left(\dfrac{\Delta t}{\Delta x^2}.h(L_x, y_1, t_n) + \dfrac{\Delta t}{\Delta y^2}.g(x_{N_x}, 0, t_n)\right) \\ f_{N_x+1}^n & + D\dfrac{\Delta t}{\Delta x^2}.h(0, y_2, t_n) \\ \vdots & \\ f_k^n & \\ \vdots & \\ f_{N.x.N_y}^n & + D\left(\dfrac{\Delta t}{\Delta x^2}.h(L_x, y_{N_y}, t_n) + \dfrac{\Delta t}{\Delta y^2}.g(x_{Nx}, L_y, t_n)\right) \end{pmatrix} \tag{4}$$

La mise en équation du problème est terminée. Le système s'écrit alors $A \times U^{n+1} = U^n + F$
A étant inversible, l'existence de U^{n+1} est assurée pour tout $n \geq 0$. Le programme aura donc pour fonction de résoudre le système de manière itérative pour n croissant.

A noter que dans le cas d'une solution stationnaire, une seule itération sera suffisante. De plus, la matrice A devra être légèrement modifiée car le terme dérivée en temps sera nul, ce qui entraîne la modification du coefficient diagonal :

$$\alpha_{stat} = 2D.\Delta t\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)$$

II Code séquentiel

II.1 Configuration du problème

Le programme se base sur la numérotation suggérée, et étudie l'un des cas tests suivants, au choix de l'utilisateur :

Cas stationnaires

$$\begin{cases} f_1(x,y) = 2(y - y^2 + x - x^2) & , \ g_1 = 0 \quad , \quad h_1 = 0 \\ f_2(x,y) = \left(\frac{4\pi}{L_x}\right)^2 sin\left(x\frac{4\pi}{L_x}\right) + \left(\frac{4\pi}{L_x}\right)^2 cos\left(y\frac{4\pi}{L_y}\right) & , g_2(x,y) = h_2(x,y) = sin\left(x\frac{4\pi}{L_x}\right) + cos\left(y\frac{4\pi}{L_y}\right) \end{cases}$$

Nous avons pris la liberté de normaliser les fonctions du cas 2 afin d'adapter la courbe au domaine. Pour ces deux premiers tests, la solution exacte se retrouve assez facilement :

$$\begin{cases} u_1(x,y) = (y^2 - y)(x^2 - x) \\ u_2(x,y) = sin\left(x\frac{4\pi}{L_x}\right) + cos\left(y\frac{4\pi}{L_y}\right) \end{cases}$$

Un calcul d'erreur pourra donc être effectué. Comme expliqué précédemment, le problème stationnaire nécessite une révision de la matrice. La variable '**nstat**' du programme principal rend compte de ce choix binaire. Si elle est nulle, la formulation stationnaire est adoptée et l'algorithme ne sera itéré qu'une unique fois. Les tests de validation seront effectués sur ces cas.

Cas instationnaire

$$f_3(x,y,t) = e^{-\left(x-\frac{L_x}{2}\right)^2} e^{-\left(y-\frac{L_y}{2}\right)^2} cos(\pi t) \quad , g_3 = 0 \quad , h_3 = 1$$

La solution analytique est cette fois-ci inconnue, et **nstat** = 1 ici. Les tests de performance présentés dans la partie suivante seront effectués sur ce cas.

Enfin si elles ne sont pas précisées dans le fichier '**parametre.dat**', les valeurs des paramètres par défaut seront les suivantes :

$$L_x = 1 \qquad L_y = 1 \qquad D = 1 \qquad T = 50$$

II.2 Exécution

Le déroulement est le suivant :

- Les paramètres $N_x, N_y, L_x, L_y, D, dx, dy, dt$ sont lus dans le fichier '**parametres.dat**'
- Le choix du cas test ainsi que d'autres éléments sont laissés à l'utilisateur dans le même fichier.
- Le second membre est alors calculé à l'aide du module **Fonctions** et la sous-routine **Init** qui permet d'obtenir la numérotation locale à partir de la globale.
- La matrice A n'est pas calculée explicitement : La sous-routine **Prod_Ap** effectue le produit A.X à partir des coefficients α, β, et γ. Elle est appelée dans la sous-routine **Gradient_conjugue** pour résoudre le système dans la boucle principale en temps.
- Les données de calcul sont relevées dans des fichiers formatés numérotés '**Sol-*.dat**' suivant les itérations en temps
- Un script Gnuplot est appelé en fin de programme pour le tracé de la solution finale ou d'une animation, suivant le choix de l'utilisateur.
- Le répertoire de travail est nettoyé en début et en fin de procédure.

II.3 Vérification Numérique

On souhaite vérifier que la solution numérique calculée avec l'algorithme est correcte. Pour les deux premiers choix, on connaît les solutions exactes (voir précédemment). On compare donc la solution numérique à la solution exacte :

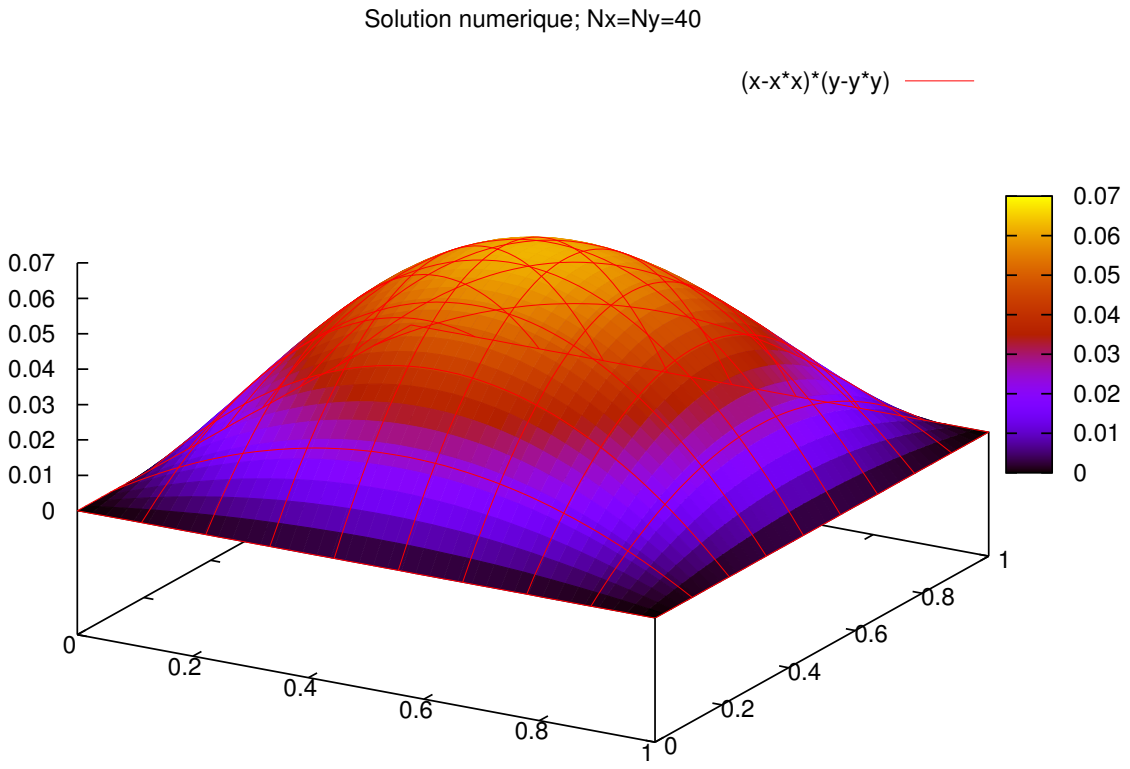


FIGURE 1 – Comparaison entre solutions numérique et exacte pour le problème 1

On observe graphiquement la superposition des solutions numérique et exacte. Un calcul d'erreur en norme 2 (différence entre solutions exacte et numérique point à point) donne la valeur $3.044E-5$, validant la convergence. La vérification est effectuée pour les deux premiers cas, avec un résultat positif.

La solution au problème 3 n'est pas de formulation explicite, mais on sait que la solution est périodique. On réalise donc une animation de la solution afin de visualiser la périodicité de celle-ci. On vérifie également que le raffinement du maillage n'influe pas sur la convergence/périodicité de la solution

REMARQUE : Si le pas d'espace est très faible (Nx, Ny élevés), un affinage du pas de temps et du critère de convergence de l'algorithme peut s'avérer nécessaire.

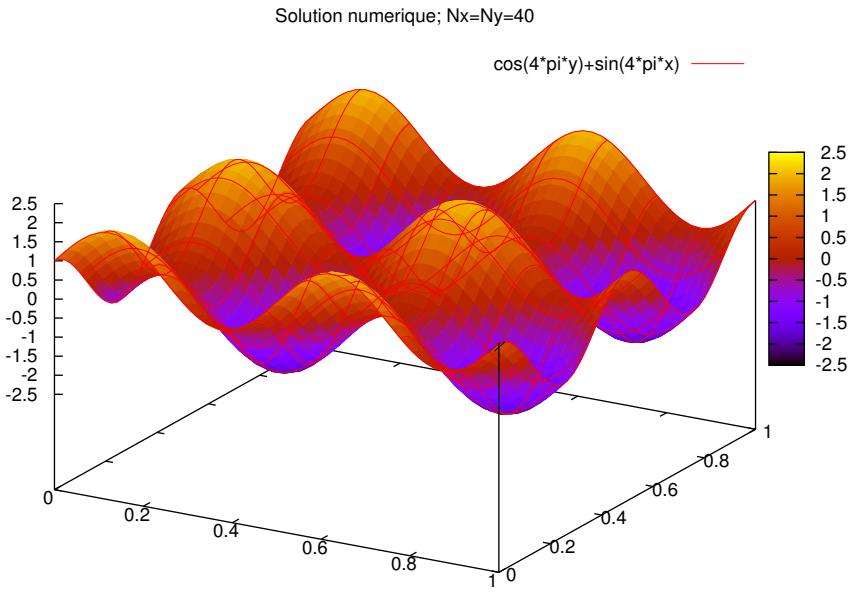


FIGURE 2 – Comparaison entre solutions numérique et exacte pour le problème 2

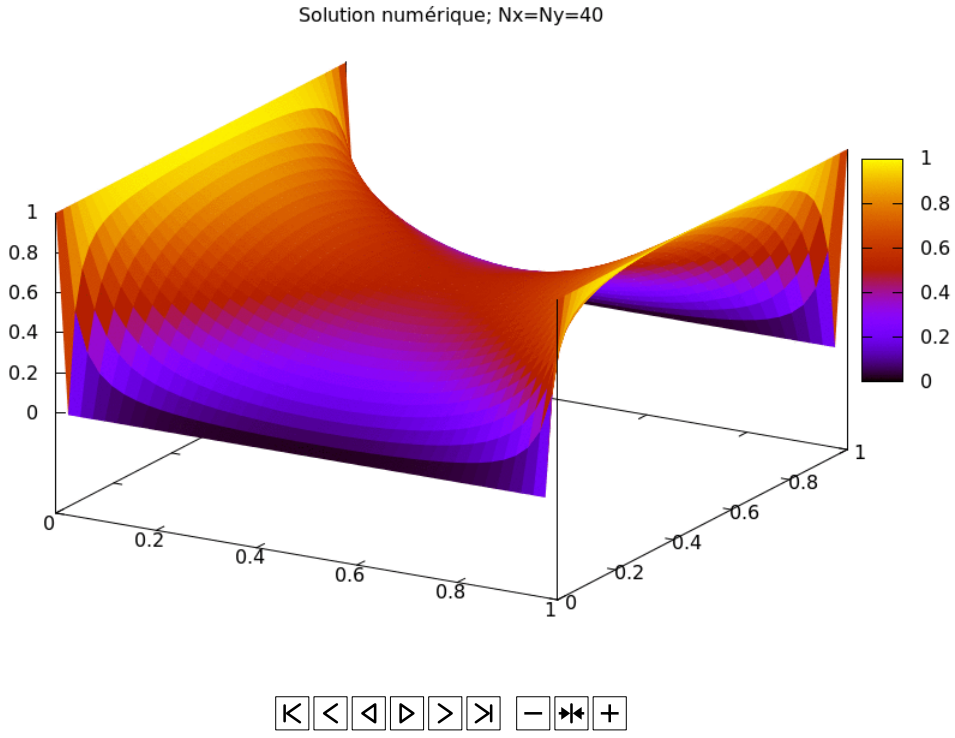


FIGURE 3 – Périodicité de la solution numérique du problème 3

III Code parallèle

III.1 Répartition des inconnues

Le vecteur U est composé de Ny paquets de Nx composantes, chacun de ces paquets correspondant à une ligne de la matrice de numérotation globale.

On remarque que pour faire le calcul de la i$^{\text{ème}}$ composante de U lors du produit matrice-vecteur, on a besoin des composantes voisines sollicitées lors du produit matrice-vecteur :

— La i$^{\text{ème}}$, pour la diagonale (coef α)
— La i$-1^{\text{ème}}$ et la i$+1^{\text{ème}}$, pour les sur/sous diagonales (coef β)
— les i$-Nx$ et i$+Nx^{\text{ème}}$, sur les $Nx^{\text{ème}}$ sur/sous-diagonale (coef γ)

Ainsi, chaque processeur aura besoin, en plus des composantes de U qu'il connaît, des Nx précédentes et des Nx suivantes.

La charge de calcul est répartie en nombre de lignes de la matrice A : Une répartition plus équitable est possible en 'coupant' les lignes avec une surcharge plus faible, mais le gain est minime si on considère que les dimensions de la matrice sont bien supérieure au nombre de processus. De plus, cette structure implique un nombre important de boucle 'if' supplémentaire ralentissant le calcul du produit A.P. Ainsi, il suffira à chaque processeur de communiquer ses Nx dernières composantes au processeur suivant, et ses Nx premières composantes au processeur précédent.

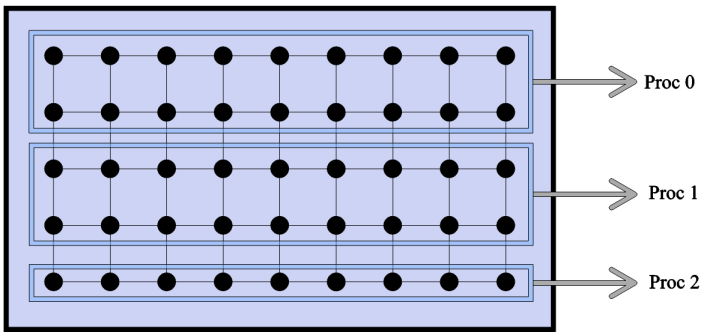


FIGURE 4 – Répartition globale des variables

En pratique, chaque processeur se voit attribuer $k_i * Nx$ composantes consécutives du vecteur U, de telle sorte que $\sum_{i=0}^{Np-1} k_i = Ny$ et $|k_i - k_j| \leq 1$.

III.2 Communications supplémentaires

Pour paralléliser l'algorithme séquentiel, il est nécessaire d'introduire plusieurs communications entre les différents processeurs :

1. Partage de la variable correspondant au choix du problème à résoudre. Un BCAST et une BARRIER. Deux autres attentes également, après l'initialisation de tous les vecteurs et avant concaténation de la solution pour le tracé sur gnuplot. Type BARRIER.
2. Dans la sous-routine du gradient conjugué, on calcule les variables intermédiaires globales pour chacun des processeurs. Un ALLREDUCE en dehors de la boucle, puis trois autres par itérations.
3. Dans la sous-routine du produit matrice-vecteur optimisé, en raison de la forme particulière de la matrice, chaque processeur a besoin des Nx composantes précédentes et des Nx composantes suivantes afin de faire le calcul $A.U$. Deux SEND et deux RECV par processeur (sauf pour le premier et le dernier où l'on en a seulement un de chaque).

III.3 Description des communications

1. Programme principal : Un entier envoyé à chacun des processeurs avec le BCAST.

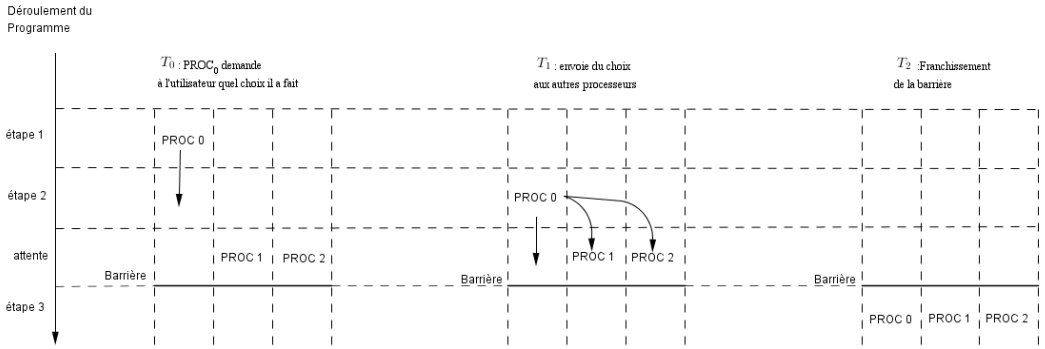


FIGURE 5 – Utilisation couplé barrière/boardcast pour la variable choix

2. Gradient Conjugué : les Np réels double précision, un par processeurs, sont récupérés et sommés à chaque ALLREDUCE, cette somme étant partagée avec tous les processeurs. Cette communication survient une fois par appel de la sous-routine, puis trois fois par itérations.

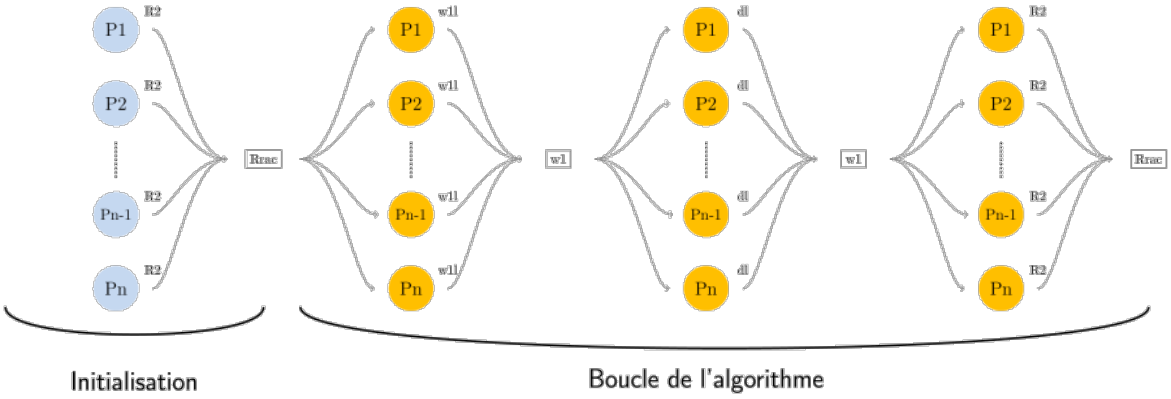


FIGURE 6 – Réductions successives - Subroutine Gradient conjugué

3. Produit matrice-vecteur : chaque processeur (sauf le premier) envoie Nx réels double précision au processeur précédent, et chaque processeur (sauf le dernier), envoie Nx réels double précision au processeur suivant. L'opération advient à chaque itération du gradient conjugué.

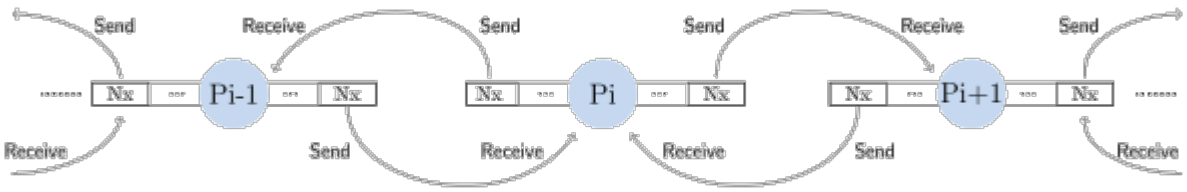


FIGURE 7 – Communication point à point : Produit Matrice/vecteur

De plus, chaque processeur va écrire, dans un fichier différent portant son numéro d'identification (Me), les composantes du vecteurs U qu'il connaît et les coordonnées correspondantes, formatées pour la compatibilité Gnuplot. Une fois l'écriture de tous les fichiers achevée, ils sont concaténés dans l'ordre dans un fichier nommé $SolTotale$. Un script visualise alors le résultat.

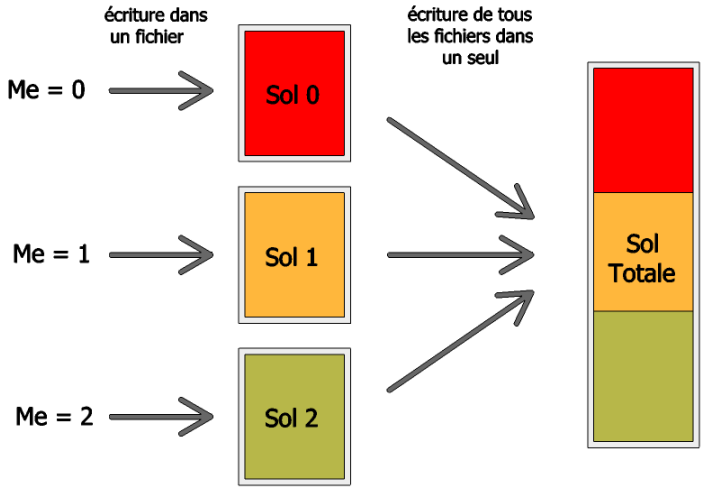


FIGURE 8 – Concaténation des fichiers de sorti

REMARQUE : La commande bash utilisé pour la concaténation est `'cat Sol-*.dat > SolFinale.dat'`. On note que les fichiers sont automatiquement fusionnés dans l'ordre des numéros croissant derrière la dénomination '`Sol-*`'

III.4 Performances du programme parallèle

REMARQUE : La version parallèle de notre programme est faite pour être exécuter sur plusieurs processeurs uniquement. Son utilisation avec un seul processeur n'introduit pas d'erreur de calcul, mais **fausse la sortie graphique**.

Le Speed-up se calcule par la formule :

$$S_p = \frac{T_1}{T_p}$$

Avec T_1 le temps de référence du programme sur un process, et T_p le temps d'exécution pour p process.

Une fois la convergence vérifiée, **on désactive toutes les sorties fichiers/graphiques** afin d'évaluer uniquement les performance en calcul. Même si en réalité, les temps d'écritures, et d'affichage graphique suivant le matériel sont loin d'être négligeables ... Le maillage et le pas de temps sont ajustés pour obtenir des temps d'exécution exploitable suffisamment grands (de l'ordre de la seconde), de variation négligeable.

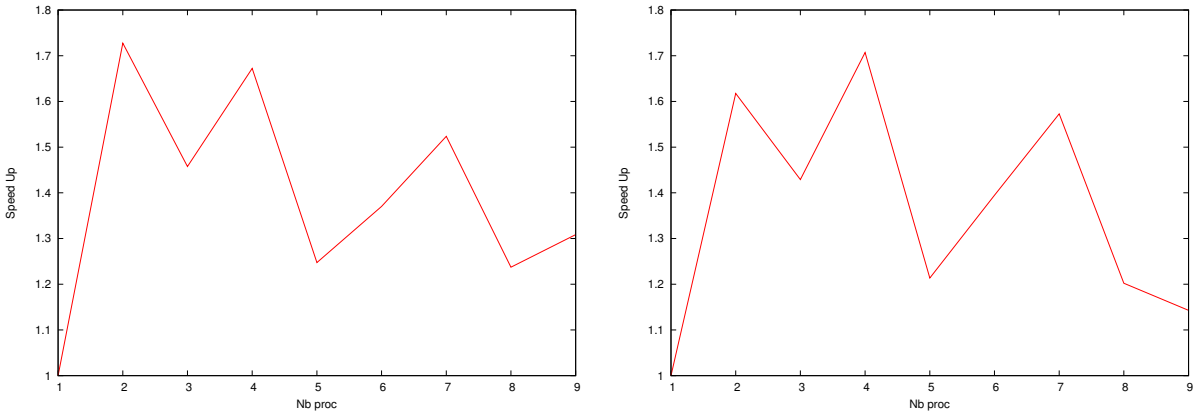
Le relevé du temps est effectué à l'aide de la fonction `MPI_WTIME()`.

REMARQUE : L'utilisation de `CPU_TIME` ne donne pas des temps physiquement cohérent en parallèle. Néanmoins, les temps affichés ne sont pas aléatoires : ils semblent tout simplement être le temps de calcul réel divisé par le nombre d'unité simulées...

Les calculs ont été effectués sur un processeur **Intel Core i5 4200H 4x2.8 GHz**.

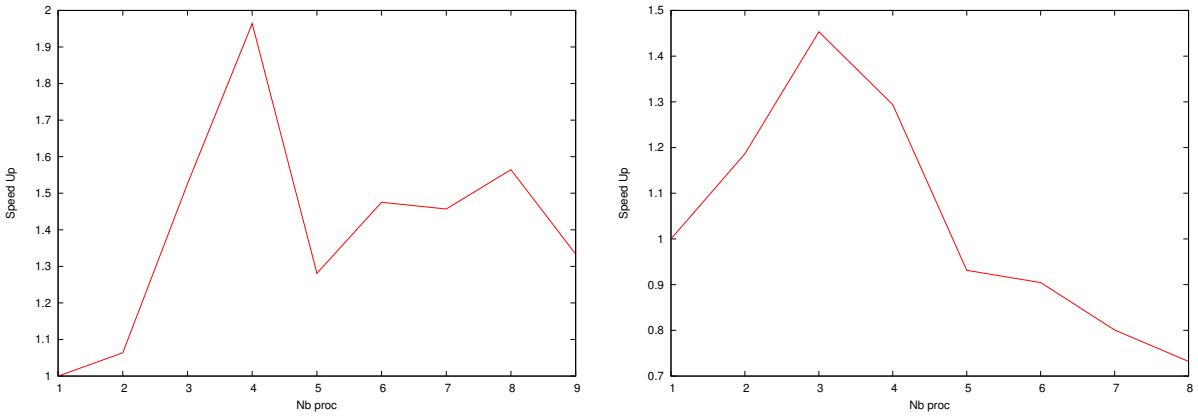
Étant donné que ce modèle n'a que 4 unités de calcul, les valeurs au delà de 4 ne sont pas très significatives. On constate néanmoins un gain de performance significatif lorsque le calcul est réparti sur 2,3 ou 4 Cœurs. De plus, les valeurs impaires induisant une surcharge, produisent des résultats moins intéressants. Enfin, on constate qu'en instationnaire, le grand nombre de communications intra process impacte sévèrement le speed-up lorsque le nombre de cœurs simulés augmente.

Le meilleur gain s'obtient donc ici pour une exécution sur 2 ou 4 processeurs.



(a) Pb. 1 (b) Pb. 2

FIGURE 9 – Speed-Up : Pb stationnaire, 300x300 points



(a) 150x150 pts, 50 itérations (b) 90x90 pts, 300 itérations

FIGURE 10 – Speed-Up : Solution stationnaire, dt = 0.1

IV Conclusion

Au long de ce projet, nous avons réalisé un programme séquentiel résolvant l'équation de la chaleur en 2D pour 3 jeux conditions aux bords, puis parallélisé ce programme. Nous avons ensuite effectué différents tests de speed-up pour déterminer le gain de performance.

Les principales difficultés lors de la parallélisation ont été la minimisation des communications entre process et l'optimisation de la répartition de charge de travail. Le gain de performance lors du passage en programme parallèle est conséquent, de l'ordre du rapport Charge totale/Nb de procs. Néanmoins, ce gain à un coût à plusieurs niveau :

Tout d'abord, au niveau de l'implémentation du code parallèle, qui demande du temps et de la réflexion. Écrire un code de ce type requiert un certains nombre de nouveau réflexes absent de la programmation 'conventionnelle'.

Ensuite, la parallélisation possède des limites : le temps de calcul dépendant également du nombres de communications entre les différents process au cours du programme, plus leur nombre est élevé, plus la durée d'exécution sera importante. Il devient fondamental d'optimiser les phases de communication en essayant au mieux de les éviter et au pire de les regrouper.

Comme expliqué dans les pages précédentes, les temps d'écriture et d'affichage ne sont également pas prix en compte dans le calcul du speed-up, faussant les véritables temps d'exécution. Un programme massivement parallèle nécessiterait donc au moins un disque rapide (SSD) pour accélérer la vitesse d'écriture.

Son dernier coût est donc ... financier. Multiplier les cœurs ne divise pas linéairement la vitesse d'exécution, et un grand nombre de processeurs peut même se révéler plus lent.

Ainsi, le calcul haute performance permet un gain en temps significatif pour les programmes ayant des charges de calcul importantes. Un compromis doit être trouvé entre le gain de temps attendu, la stratégie de parallélisation, le temps à consacré au développement et le budget disponible. Les limitations sont nombreuses et doivent être pris en compte toute au long de la programmation pour le meilleur résultat.

On peut également affirmer qu'il est nécessaire de développer dans un premier temps une version séquentielle la plus propre possible afin de pouvoir programmer la version parallèle correctement.

A Voir également : HPC is dying and MPI is killing it