

Calcul Haute Performance et Simulation

Présentation d'un projet

**Analyse numérique d'équations aux dérivées partielles par différences finies
et implémentation optimisée pour le calcul haute performance**

par Jean-Baptiste Gaillot

version 1.1

Cette présentation est la suite d'une présentation initiale réalisé dans le cadre d'un stage du Master Calcul Haute Performance et Simulation (1ère année, 2ème semestre à l'Université de Perpignan Via Domitia pour l'année universitaire 2024–2025) comportant quelques améliorations réalisées à la suite de ce dernier.

Approche :

- concevoir un ou plusieurs schémas numériques pour obtenir une solution approchée du problème,
- s'assurer de l'existence et de l'unicité de la solution approchée,
- s'assurer des bonnes propriétés du schéma (consistance, convergence et erreur locale),
- concevoir un ou plusieurs schémas de résolution de l'éventuel système linéaire associé à ce schéma.

Organisation – Partie informatique

Approche :

- implémenter des fonctions de résolutions du problème et un programme principal,
- implémenter le calcul d'une solution exacte connue dans le but de calculer l'erreur entre la solution approchée et la solution exacte,
- implémenter la résolution du problème en différentes versions dans le but de comparer les performances (différents schémas en versions naïves, séquentielles, parallèles et utilisation d'une bibliothèque),
- structurer toutes ces étapes à travers un projet.

Les différents résultats (erreurs et temps d'exécutions) seront présentés sous forme de tableaux et de graphiques. Le langage de programmation utilisé est C.

Problèmes étudiés

Équation de Poisson en dimension 1

Équation de Poisson en dimension 2

Équation des ondes en dimension 1

Équation de la chaleur en dimension 2

Équation de Poisson en dimension 1

Problème

Soient $D :=]0, 1[$, $f : D \rightarrow \mathbb{R}$ continue et bornée et le problème suivant :

Trouver u de classe C^4 telle que :

$$\begin{cases} -u''(x) = f(x) & \forall x \in D \\ u(x) = 0 & \forall x \in \partial D \end{cases}.$$

Discrétisation

$$-u''(x) = \frac{1}{h^2} (-u(x+h) + 2u(x) - u(x-h)) + E_h$$

avec

$$E_h := \frac{1}{12} h^2 u^{(4)}(x + \theta h).$$

Schéma

$$\boxed{\frac{1}{h^2} (-u_{i+1} + 2u_i - u_{i-1}) = f_i.}$$

Schéma sous forme matricielle

$$\boxed{Au = f}$$
$$\Leftrightarrow \underbrace{\frac{1}{h^2} \begin{pmatrix} 2 & -1 & \cdot & \cdot \\ -1 & \ddots & \ddots & \cdot \\ \cdot & \ddots & \ddots & -1 \\ \cdot & \cdot & -1 & 2 \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_1 \\ \vdots \\ \vdots \\ u_{N-1} \end{pmatrix}}_{=u} = \underbrace{\begin{pmatrix} f_1 \\ \vdots \\ \vdots \\ f_{N-1} \end{pmatrix}}_{=f}.$$

Équation de Poisson en dimension 1 – Analyse numérique

Existence et unicité de la solution approchée

Proposition A est définie-positive et $Au = f$ admet une unique solution.

Équation de Poisson en dimension 1 – Analyse numérique

Consistance du schéma et majoration de l'erreur de troncature

Proposition Le schéma est consistant : $\lim_{h \rightarrow 0} |E_h| = 0$ et

$$|E_h| \leq \frac{1}{12} h^2 \sup_{x \in [0,1]} |f''(x)|.$$

Remarque Le schéma est d'ordre 2 pour x .

Proposition Soient $h > 0$, $e_h := (u_i - u(x_i))_{0 \leq i \leq N}$. Alors, le schéma est convergent :

$$\lim_{h \rightarrow 0} \|e_h\|_{\infty} = 0 \quad \text{et} \quad \|e_h\|_{\infty} \leq \frac{1}{96} h^2 \sup_{x \in [0,1]} |f''(x)|.$$

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution itérative

Méthode de Jacobi

$$Du^{(k+1)} = (E + F) u^{(k)} + f.$$

Schéma

$$u_i^{(k+1)} = \frac{1}{2} \left(u_{i-1}^{(k)} + u_{i+1}^{(k)} + h^2 f_i \right).$$

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution directe

Factorisation de Cholesky

$$\ell_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} \ell_{i,k}^2} \quad \text{et} \quad \ell_{i,j} = \frac{1}{\ell_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k} \ell_{j,k} \right).$$

On calcule d'abord en colonnes puis en lignes.

Proposition Soit A une matrice tridiagonale, symétrique et définie-positive. Alors, la matrice L de la décomposition de Cholesky de A est bidiagonale inférieure.

Schéma

$$\text{Pour } j \text{ de } 1 \text{ à } N-1 : \begin{cases} \ell_{i,i} = \sqrt{\alpha - \ell_{i,i-1}^2} & \text{si } d = 0 \\ \ell_{i,j} = \frac{\beta}{\ell_{j,j}} & \text{si } d = 1 \end{cases}.$$

avec

$$d := i - j, \quad \alpha := \frac{2}{h^2} \quad \text{et} \quad \beta := -\frac{1}{h^2}.$$

Équation de Poisson en dimension 1 – Analyse numérique

Méthode de résolution directe

Schéma

$$y_1 = \frac{f_1}{\ell_{1,1}} \quad \text{et} \quad \text{pour } i \text{ de } 2 \text{ à } N-1 : y_i = \frac{f_i - \ell_{i,j-1}y_{i-1}}{\ell_{i,i}}$$

et

$$u_{N-1} = \frac{y_{N-1}}{\ell_{N-1,N-1}} \quad \text{et} \quad \text{pour } i \text{ de } N-2 \text{ à } 1 : u_i = \frac{y_i - \ell_{i,j+1}u_{i+1}}{\ell_{i,i}}.$$

Pour la suite, la fonction à approcher sera avec $f(x) = \pi^2 \sin(\pi x)$.

Équation de Poisson en dimension 1 – Implémentation

Version de base

Commentaires

- Pour calculer u , on résout le système linéaire avec la méthode de Gauss.
- On note ces résultats :

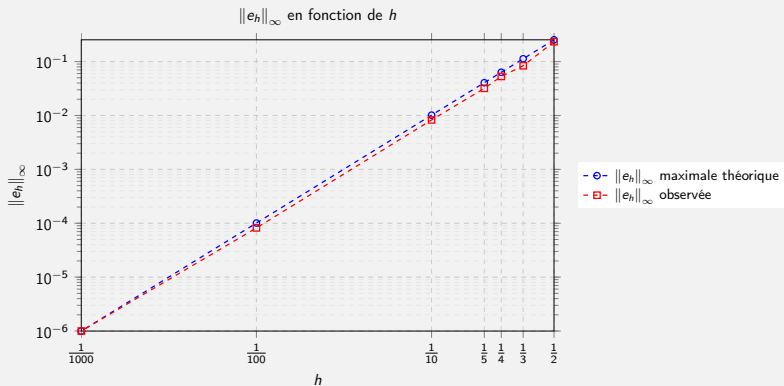
N	5	10	50	100	300	500	1500
$\ e_h\ _\infty$	0.031916	0.008265	0.000329	0.000082	0.000009	0.000003	<0.000001
Tps d'ex. (s)	<0.01	<0.01	<0.01	<0.01	0.01	0.05	1.00

- A est de taille $O(N)$ et la méthode de Gauss est $O(N^3)$ donc la complexité algorithmique est $O(N^3)$.

Équation de Poisson en dimension 1 – Implémentation

Version de base

Commentaire On vérifie la proposition énoncée avec $f(x) = \pi^2 \sin(\pi x)$, d'après la proposition : $\|e_h\|_\infty \leq \frac{1}{96} \pi^4 h^2$.



Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Rappel du schéma :

$$u_i^{(k+1)} = \frac{1}{2} \left(u_{i-1}^{(k)} + u_{i+1}^{(k)} + h^2 f_i \right).$$

Fontion qui applique le schéma à un nœud :

```
static inline __attribute__((always_inline))
double schema(double *f, double *u_anc, int i){
    double res = 0.5 * ((u_anc[i - 1] + u_anc[i + 1]) + h_carre * f[i]);
    return res;
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Commentaire On note ces résultats :

N	5	10	50	100	300	500	1500
Nb. d'itérations	102	400	8506	31227	241002	617699	4557543
$\ e_h\ _\infty$	0.031916	0.008265	0.000329	0.000082	0.000007	0.000002	0.000045
Tps d'ex. (s)	<0.01	<0.01	<0.01	0.01	0.04	0.20	4.67

Équation de Poisson en dimension 1 – Implémentation

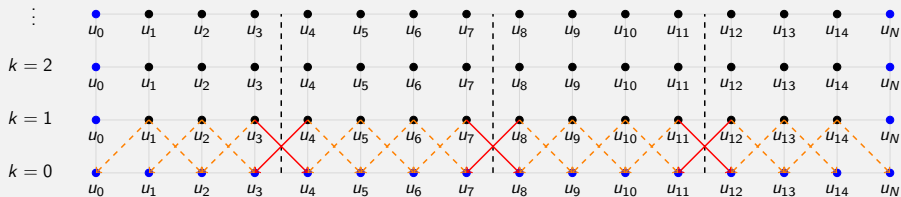
Version avec méthode de résolution itérative en parallèle avec OpenMP

Commentaire On ajoute une directive `for` dans la boucle de la fonction `calculer_u_jacobi` et une directive `for` dans la boucle du calcul de la norme relative.

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Illustration Schéma des dépendances pour 4 processus et $N = 15$:



Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonctions pour MPI

```
void creer_topologie(){  
    int tore = 0;  
    dims = 0;  
  
    MPI_Dims_create(nb_cpu, 1, &dims);  
  
    MPI_Cart_create(MPI_COMM_WORLD, 1, &dims, &tore, 0, &comm_1D);  
  
    MPI_Barrier(comm_1D);  
}
```

```
void infos_processus(){  
  
    i_debut = (coords * nb_pt) / dims;  
    i_fin = ((coords + 1) * nb_pt) / dims - 1;  
    nb_pt_div = i_fin - i_debut + 1;  
  
    MPI_Barrier(comm_1D);  
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonction pour obtenir les indices de départ et d'arrivé de la boucle principale :

```
void infos_bornes_boucles(int *i_boucle_debut, int *i_boucle_fin){  
    *i_boucle_debut = 1;  
    *i_boucle_fin = nb_pt_div + 1;  
  
    if (i_debut == 0){  
        (*i_boucle_debut) ++;  
    }  
    if (i_fin == nb_pt - 1){  
        (*i_boucle_fin) --;  
    }  
}
```

Fonction pour échanger les halos :

```
void echanger_halos(double *u_div){  
    // Envoi gauche, reception droite  
    MPI_Sendrecv  
    (&(u_div[1]), 1, MPI_DOUBLE, voisins[0], etiquette, &(u_div[nb_pt_div + 1]),  
    1, MPI_DOUBLE, voisins[1], etiquette, comm_1D, &statut);  
  
    // Envoi droite, reception gauche  
    MPI_Sendrecv  
    (&(u_div[nb_pt_div]), 1, MPI_DOUBLE, voisins[1], etiquette, &(u_div[0]), 1, MPI_DOUBLE,  
    voisins[0], etiquette, comm_1D, &statut);  
}
```


Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe

Structure `mat_2bandes` :

```
struct mat_2bandes{
    int N;
    double *diag; // taille N - 1
    double *sous_diag; // taille N - 2
};
```

Fonction pour allouer la structure :

```
void init_mat_2bandes(struct mat_2bandes *A){
    A -> N = N;
    A -> diag = (double *)malloc(idx_max * sizeof(double));
    A -> sous_diag = (double *)malloc((idx_max - 1) * sizeof(double));
}
```

Fonction pour libérer la structure :

```
void liberer_mat_2bandes(struct mat_2bandes *A){
    free(A -> diag);
    free(A -> sous_diag);
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe

Rappel du schéma :

$$\text{Pour } j \text{ de } 1 \text{ à } N-1 : \begin{cases} \ell_{i,j} = \sqrt{\alpha - \ell_{i,i-1}^2} & \text{si } d = 0 \\ \ell_{i,j} = \frac{\beta}{\ell_{j,j}} & \text{si } d = 1 \end{cases}$$

Fonction pour obtenir la décomposition de Cholesky :

```
void calculer_cholesky(struct mat_2bandes *L){
    h_carre = 1.0 / pow(N, 2);
    double alpha = 2.0 / h_carre;
    double beta = -1.0 / h_carre;

    (L -> diag)[0] = sqrt(alpha);
    (L -> sous_diag)[0] = beta / (L -> diag)[0];

    for (int i = 1 ; i < idx_max - 1 ; i ++){
        (L -> diag)[i] = sqrt(alpha - pow((L -> sous_diag[i - 1]), 2));
        (L -> sous_diag)[i] = beta / (L -> diag[i]);
    }

    (L -> diag)[idx_max - 1]
    = sqrt(alpha - pow((L -> sous_diag[idx_max - 2]), 2));
}
```

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe

Test pour avoir un aperçu de la compression

Illustration de la structure mat_2bandes (exemple pour N petit) :

Structure mat2_bandes :

N = 7

diag	=	9.899495	8.573214	8.082904	7.826238	7.668116	7.560864
sous_diag	=	-4.949747	-5.715476	-6.062178	-6.260990	-6.390097	

Matrice reelle correspondante :

9.899495	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
-4.949747	8.573214	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	-5.715476	8.082904	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	-6.062178	7.826238	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	-6.260990	7.668116	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	-6.390097	7.560864	

Équation de Poisson en dimension 1 – Implémentation

Version avec méthode de résolution directe

Fonction pour résoudre $Ly = f$:

```
void resoudre_cholesky_descente(struct mat_2bandes *L, double *f, double *y){
    y[0] = f[0] / (L -> diag)[0];
    for (int i = 1 ; i < idx_max ; i++){
        y[i] = (f[i] - (L -> sous_diag)[i - 1] * y[i - 1]) / (L -> diag)[i];
    }
}
```

Rappel du schéma :

$$u_{N-1} = \frac{y_{N-1}}{\ell_{N-1,N-1}} \quad \text{et} \quad \text{pour } i \text{ de } N-2 \text{ à } 1 : u_i = \frac{y_i - \ell_{i,j+1}u_{j+1}}{\ell_{i,i}}.$$

Fonction pour résoudre $L^T u = y$:

```
void resoudre_cholesky_remontee(struct mat_2bandes *L, double *y, double *u){
    u[idx_max - 1] = y[idx_max - 1] / (L -> diag)[idx_max - 1];
    for (int i = idx_max - 2 ; i >= 0 ; i--){
        u[i] = (y[i] - (L -> sous_diag)[i] * u[i + 1]) / (L -> diag)[i];
    }
}
```

Équation de Poisson en dimension 1 – Implémentation

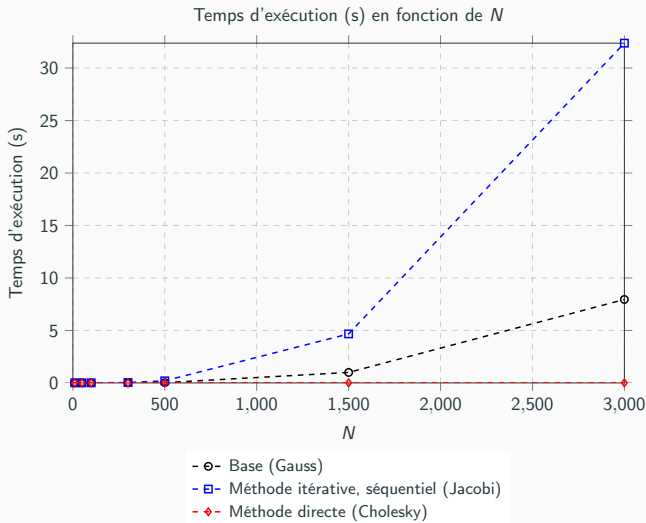
Version avec méthode de résolution directe

Commentaires

- Le temps d'exécution est < 0.01 s pour $N = 1000000$.
- A possède $O(N)$ colonne. Pour chaque colonne, il y a $O(1)$ lignes à calculer. Pour chaque case, il y a $O(1)$ opérations. Donc la complexité algorithmique est $O(N)$.

Équation de Poisson en dimension 1 – Implémentation

Comparaison des performances des méthodes



Équation de Poisson en dimension 2

Problème

Soient $D :=]0, 1[\times]0, 1[$, $f : D \rightarrow \mathbb{R}$ continue et bornée et le problème suivant :

Trouver u de classe C^4 telle que :

$$\begin{cases} -\Delta u(x, y) = f(x, y) & \forall (x, y) \in D \\ u(x, y) = 0 & \forall (x, y) \in \partial D \end{cases}.$$

Discrétisation

$$-\Delta u(x, y) = \frac{1}{h_x^2} \delta_x^2 + \frac{1}{h_y^2} \delta_y^2 + E_h$$

avec

$$\delta_x^2 := -u(x - h_x, y) + 2u(x, y) - u(x + h_x, y),$$

$$\delta_y^2 := -u(x, y - h_y) + 2u(x, y) - u(x, y + h_y)$$

et

$$E_{h_x, h_y} := \frac{1}{12} \left(h_x^2 \frac{\partial^4 u}{\partial x^4}(x + \theta_x h_x, y) + h_y^2 \frac{\partial^4 u}{\partial y^4}(x, y + \theta_y h_y) \right).$$

Schéma

$$\frac{1}{h^2} (-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1}) = f_{i,j}.$$

Schéma sous forme matricielle par blocs

$$\boxed{Au = f} \Leftrightarrow \frac{1}{h^2} \underbrace{\begin{pmatrix} \boxed{M} & \boxed{-I} & . & . \\ \boxed{-I} & . & . & . \\ . & . & . & \boxed{-I} \\ . & . & \boxed{-I} & \boxed{M} \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_1 \\ \vdots \\ \vdots \\ u_{N-1} \end{pmatrix}}_{=:u} = \underbrace{\begin{pmatrix} f_1 \\ \vdots \\ \vdots \\ f_{N-1} \end{pmatrix}}_{=:f}$$

avec

$$M := \begin{pmatrix} 4 & -1 & . & . \\ -1 & . & . & . \\ . & . & . & -1 \\ . & . & -1 & 4 \end{pmatrix}.$$

Équation de Poisson en dimension 2 – Analyse numérique

Existence et unicité de la solution approchée

Proposition A est définie-positive et $Au = f$ admet une unique solution.

Équation de Poisson en dimension 2 – Analyse numérique

Consistance du schéma et majoration de l'erreur de troncature

Notation Soit $d \in \{1, \dots, 4\}$. Alors,

$$C_{u,d} := \max \left\{ \sup_{(x,y) \in [0,1]^2} \left| \frac{\partial^d u}{\partial x^d}(x,y) \right|, \sup_{(x,y) \in [0,1]^2} \left| \frac{\partial^d u}{\partial y^d}(x,y) \right| \right\}.$$

Proposition Le schéma est consistant : $\lim_{h \rightarrow 0} |E_h| = 0$ et $|E_h| \leq \frac{1}{6} h^2 |C_{u,4}|$.

Remarque Le schéma est d'ordre 2 pour x et pour y .

Proposition (*admise*) Soient $h > 0$, $e := e_h$, $e_j := (\|u_j - u(x_j)\|_\infty)_{0 \leq j \leq N}$. Alors, le schéma utilisé est convergent :

$$\forall j \in \{1, \dots, N-1\} : \lim_{h \rightarrow 0} \|e_j\|_\infty = 0$$

et

$$\exists C > 0, \forall j \in \{1, \dots, N-1\} : \|e_j\|_\infty \leq Ch^2 (C_{u,4} + hC_{u,3}).$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution itérative

Méthode de Jacobi

$$Du^{(k+1)} = (E + F) u^{(k)} + f.$$

Schéma

$$u_{i,j}^{(k+1)} = \frac{1}{4} \left(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + h^2 f_{i,j} \right).$$

Factorisation de Cholesky

$$\ell_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} \ell_{i,k}^2} \quad \text{et} \quad \ell_{i,j} = \frac{1}{\ell_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k} \ell_{j,k} \right).$$

On calcule d'abord en colonnes puis en lignes.

Proposition Soit A une matrice avec N diagonales inférieures, symétrique et définie-positive. Alors, la matrice L de la décomposition de Cholesky de A possède N diagonales inférieures.

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution directe

Schéma

$$\text{Pour } j \text{ de } 1 \text{ à } (N-1)^2 : \begin{cases} \ell_{i,i} = \sqrt{\alpha - \sum_{k=\max\{1,j-N+d+1\}}^{i-1} \ell_{i,k}^2} & \text{si } d = 0 \\ \ell_{i,j} = \left(a_{i,j} - \sum_{k=\max\{1,j-N+d+1\}}^{j-1} \ell_{i,k} \ell_{j,k} \right) / \ell_{j,j} & \text{si } d > 0. \end{cases}$$

avec

$$d := i - j, \quad \text{et} \quad \alpha := \frac{4}{h^2}.$$

Équation de Poisson en dimension 2 – Analyse numérique

Méthode de résolution directe

Schéma

$$y_1 = \frac{f_1}{\ell_{1,1}} \quad \text{et} \quad \text{pour } i \text{ de } 2 \text{ à } (N-1)^2 : y_i = \left(f_i - \sum_{k=\max\{1, i-N+1\}}^{i-1} \ell_{i,k} y_k \right) / \ell_{i,i}$$

et

$$u_{(N-1)^2} = \frac{y_{(N-1)^2}}{\ell_{(N-1)^2, (N-1)^2}} \quad \text{et} \quad \text{pour } i \text{ de } (N-1)^2 - 1 \text{ à } 1 : u_i = \left(y_i - \sum_{k=i+1}^{\min\{i+N-1, (N-1)^2\}} \ell_{k,i} u_k \right) / \ell_{i,i} .$$

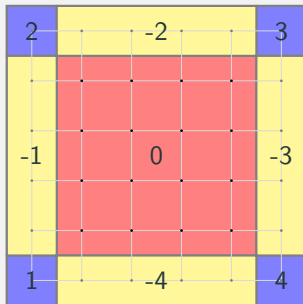
Pour la suite, la fonction à approcher sera avec $f(x, y) = \sin(2\pi x) \sin(2\pi y)$.

Équation de Poisson en dimension 2 – Implémentation

Version de base

Commentaires

- Tout les tableaux utilisés sont linéarisés pour garantir la contiguité.
- Pour calculer u , on résout le système linéaire avec la méthode de Gauss.
- Les numéros du type de bord de la fonction `connaitre_bord` sont les suivants :



Équation de Poisson en dimension 2 – Implémentation

Version de base

Commentaires (suite)

- On note ces résultats :

N	10	50	100
$\ e_h\ _\infty$	0.00038444	0.00001661	0.00000417
Tps d'ex. (s)	<0.1	4.1	278.9

- A est de taille $O(N^2)$ et la méthode de Gauss est $O(N^3)$ donc la complexité algorithmique est $O(N^6)$.

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Rappel du schéma :

$$u_{i,j}^{(k+1)} = \frac{1}{4} \left(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + h^2 f_{i,j} \right).$$

Fonction qui applique le schéma à un nœud :

```
static inline __attribute__((always_inline))
double schema(double *f, double *u_anc, int i, int j){
    double res
    = 0.25
    * (u_anc[IDX(i - 1, j)]
    + u_anc[IDX(i, j - 1)]
    + u_anc[IDX(i + 1, j)]
    + u_anc[IDX(i, j + 1)]
    + h_carre * f[IDX(i, j)]);
    return res;
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en séquentiel

Commentaire On note ces résultats :

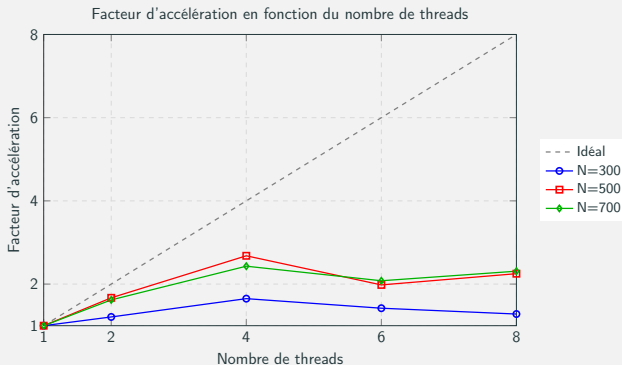
N	10	50	100	300	500	700
Nb. d'itérations	102	2298	8506	66569	171980	320379
$\ e_h\ _\infty$	0.00038444	0.00001661	0.00000417	0.00000046	0.00000015	0.00000005
Tps d'ex. (s)	<0.1	<0.1	0.1	4.8	34.6	128.4

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec OpenMP

Commentaires

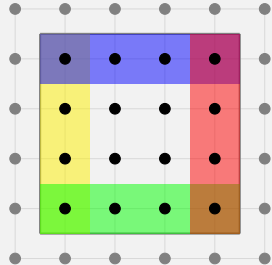
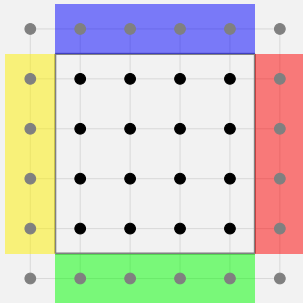
- On ajoute des directives `for`.
- On note ces résultats :



Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Illustration Schéma de la structure d'un sous-tableau :



Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Fonctions MPI

```
void creer_types(){  
    int taille_send[2] = {nb_pt_div_j + 2, nb_pt_div_i + 2};  
    int sous_taille_send[2] = {nb_pt_div_j, nb_pt_div_i};  
    int debut_send[2] = {1, 1};  
  
    MPI_Type_contiguous(nb_pt_div_i, MPI_DOUBLE, &ligne);  
    MPI_Type_vector(nb_pt_div_j, 1, nb_pt_div_i + 2, MPI_DOUBLE, &colonne);  
  
    MPI_Type_create_subarray  
    (2, taille_send, sous_taille_send, debut_send, MPI_ORDER_C, MPI_DOUBLE,  
    &bloc_send);  
  
    MPI_Type_commit(&ligne);  
    MPI_Type_commit(&colonne);  
    MPI_Type_commit(&bloc_send);  
  
    MPI_Barrier(comm_2D);  
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

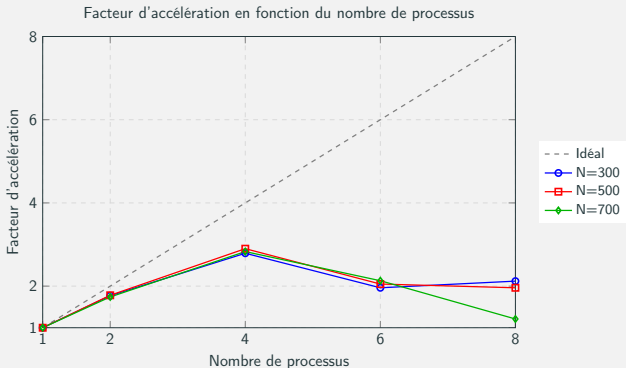
```
void echanger_halos(double *u_div){  
    // Envoi haut, reception bas  
    MPI_Sendrecv  
    (&(u_div[IDX(1, nb_pt_div_j)]), 1, ligne, voisins[1], etiquette,  
    &(u_div[IDX(1, 0)]), 1, ligne, voisins[3], etiquette, comm_2D, &statut);  
  
    // Envoi bas, reception haut  
    MPI_Sendrecv  
    (&(u_div[IDX(1, 1)]), 1, ligne, voisins[3], etiquette,  
    &(u_div[IDX(1, nb_pt_div_j + 1)]), 1, ligne, voisins[1], etiquette,  
    comm_2D, &statut);  
  
    // Envoi gauche, reception droite  
    MPI_Sendrecv  
    (&(u_div[IDX(1, 1)]), 1, colonne, voisins[0], etiquette,  
    &(u_div[IDX(nb_pt_div_i + 1, 1)]), 1, colonne, voisins[2], etiquette,  
    comm_2D, &statut);  
  
    // Envoi droite, reception gauche  
    MPI_Sendrecv  
    (&(u_div[IDX(nb_pt_div_i, 1)]), 1, colonne, voisins[2], etiquette,  
    &(u_div[IDX(0, 1)]), 1, colonne, voisins[0], etiquette, comm_2D,  
    &statut);  
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Commentaires

- Pour regrouper les résultats sur le rang 0, on utilise un type dérivé `bloc_recv` créée dynamiquement par le rang 0 (voir la fonction `regrouper_u`).
- On note ces résultats :



Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution itérative en parallèle avec MPI

Version avec un mode de communication non bloquant

Commentaires

- Dès que la communication est lancée, on fait les calculs sur l'intérieur du sous-domaine (en excluant les bords locaux), ensuite on vérifie / attend que la communication soit terminée, enfin on fait les calculs sur les bords locaux avec la fonction `test_fin_echange_halos`.
- Pour calculer sur les bords du sous-domaine (2 lignes (sauf 2 coins), 2 colonnes (sauf 2 coins) et 4 coins), on utilise la fonction `calculer_u_jacobi_bords`.

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Structure mat_Nbandes :

```
struct mat_Nbandes{
    int N;
    double **diags;
};
```

Fonction pour allouer la structure :

```
void init_mat_Nbandes(struct mat_Nbandes *A){
    A -> N = N;
    A -> diags = (double **)malloc(N * sizeof(double *));
    for (int i = 0 ; i < A -> N ; i++){
        (A -> diags)[i] = (double *)malloc((idx_max - i) * sizeof(double));
    }
}
```

Fonction pour libérer la structure :

```
void liberer_mat_Nbandes(struct mat_Nbandes *A){
    for (int i = 0 ; i < A -> N ; i++){
        free((A -> diags)[i]);
    }
    free(A -> diags);
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Rappel du schéma :

$$\left\{ \begin{array}{ll} \ell_{i,i} = \sqrt{\alpha - \sum_{k=\max\{1, j-N+d+1\}}^{i-1} \ell_{i,k}^2} & \text{si } d = 0 \\ \ell_{i,j} = \left(a_{i,j} - \sum_{k=\max\{1, j-N+d+1\}}^{j-1} \ell_{i,k} \ell_{j,k} \right) / \ell_{j,j} & \text{si } d > 0. \end{array} \right.$$

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour obtenir la décomposition de Cholesky :

```
void calculer_cholesky(struct mat_Nbandes *L){  
    h_carre = 1.0 / pow(N, 2);  
    double alpha = 4.0 / h_carre;  
  
    for (int j = 0 ; j < idx_max ; j ++){  
        for (int d = 0 ; d < N && j + d < idx_max ; d ++){  
            int i = d + j;  
  
            if (d == 0){  
                (L -> diags)[0][j] = alpha;  
                for (int k = max(0, j - N + d + 1) ; k < i ; k ++){  
                    int d_1 = i - k;  
                    (L -> diags)[0][j] -= pow((L -> diags)[d_1][k], 2);  
                }  
                (L -> diags)[0][j] = sqrt((L -> diags)[0][j]);  
            }  
  
            else{  
                (L -> diags)[d][j] = valeur_a(i, j);  
                for (int k = max(0, j - N + d + 1) ; k < j ; k ++){  
                    int d_1 = i - k;  
                    int d_2 = j - k;  
                    (L -> diags)[d][j] -= (L -> diags)[d_1][k] * (L -> diags)[d_2][k];  
                }  
                (L -> diags)[d][j] /= (L -> diags)[0][j];  
            }  
        }  
    }  
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Test pour avoir un aperçu de la compression

Illustration de la structure mat_Nbandes (exemple pour N petit) :

Structure mat_Nbandes :

N = 4

```
diag[0] = 8.0000  7.7460  7.7287  7.7275  7.3829  7.3668  7.6995  7.3261  7.3139
diag[1] = -2.0000 -2.0656 -0.1380 -2.2184 -2.3331 -0.2074 -2.2717 -2.4056
diag[2] = 0.0000 -0.5164 -0.5521 -0.0370 -0.6222 -0.6863 -0.0585
diag[3] = -2.0000 -2.0656 -2.0702 -2.0705 -2.1672 -2.1719
```

Matrice réelle correspondante :

```
8.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
-2.0000  7.7460  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
0.0000 -2.0656  7.7287  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
-2.0000 -0.5164 -0.1380  7.7275  0.0000  0.0000  0.0000  0.0000  0.0000
0.0000 -2.0656 -0.5521 -2.2184  7.3829  0.0000  0.0000  0.0000  0.0000
0.0000  0.0000 -2.0702 -0.0370 -2.3331  7.3668  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000 -2.0705 -0.6222 -0.2074  7.6995  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000 -2.1672 -0.6863 -2.2717  7.3261  0.0000
0.0000  0.0000  0.0000  0.0000  0.0000 -2.1719 -0.0585 -2.4056  7.3139
```


Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction pour résoudre $Ly = f$:

```
void resoudre_cholesky_descente(struct mat_Nbandes *L, double *f, double *y){
    y[0] = f[0] / (L -> diags)[0][0];
    for (int i = 1 ; i < idx_max ; i ++){
        y[i] = f[i];
        for (int k = max(0, i - N + 1) ; k < i ; k ++){
            int d = i - k;
            y[i] -= (L -> diags)[d][k] * y[k];
        }
        y[i] /= (L -> diags)[0][i];
    }
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Rappel du schéma :

$$u_{(N-1)^2} = \frac{y_{(N-1)^2}}{\ell_{(N-1)^2, (N-1)^2}} \quad \text{et} \quad \text{pour } i \text{ de } (N-1)^2 - 1 \text{ à } 1 : u_i = \left(y_i - \sum_{k=i+1}^{\min\{i+N-1, (N-1)^2\}} \ell_{k,i} u_k \right) / \ell_{i,i}.$$

Fonction pour résoudre $L^T u = y$:

```
void résoudre_cholesky_remontee(struct mat_Nbandes *L, double *y, double *u){
    u[idx_max - 1] = y[idx_max - 1] / (L -> diags)[0][idx_max - 1];
    for (int i = idx_max - 2 ; i >= 0 ; i --){
        u[i] = y[i];
        for (int k = i + 1 ; k < min(i + N, idx_max) ; k ++){
            int d = k - i;
            u[i] -= (L -> diags)[d][i] * u[k];
        }
        u[i] /= (L -> diags)[0][i];
    }
}
```

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Commentaires

- On note ces résultats :

N	10	50	100	300	500	700
$\ e_h\ _\infty$	0.00038444	0.00001661	0.00000417	0.00000046	0.00000017	0.00000009
Tps d'ex. (s)	<0.1	<0.1	<0.1	4.6	35.8	383.5

- A possède $O(N^2)$ colonne. Pour chaque colonne, il y a $O(N)$ lignes à calculer. Pour chaque case, il y a $O(N)$ opérations. Donc la complexité algorithmique est $O(N^4)$.

Bibliothèque Cholmod

Étapes :

- créer une fonction pour définir la structure de matrice creuse en créant des tableaux :
 - `lignes` qui contient les indices des lignes où se trouve une valeur non nulle,
 - `valeurs` qui contient les valeurs aux indices stockés,
 - `offsets` qui contient le nombre de valeurs non nulles d'une colonne,(voir les fonctions `construire_matrice_creuse` et `connaitre_bord`).
- créer une fonction qui fait le travail

Équation de Poisson en dimension 2 – Implémentation

Version avec méthode de résolution directe en séquentiel

Fonction principale :

```
void resoudre(cholmod_sparse *A, double *f, double *u){
    h_carre = 1.0 / pow(N, 2);
    double *f_int = (double *)malloc(idx_max * sizeof(double));
    double *u_int = (double *)malloc(idx_max * sizeof(double));

    extraire_interieur(f, f_int, nb_pt);
    extraire_interieur(u, u_int, nb_pt);

    cholmod_dense *f_dense =
    cholmod_allocate_dense(A -> nrow, 1, A -> nrow, CHOLMOD_REAL, &c);
    memcpy(f_dense -> x, f_int, A -> nrow * sizeof(double));

    cholmod_factor *L = cholmod_analyze(A, &c);
    cholmod_factorize(A, L, &c);

    cholmod_dense *u_dense = cholmod_solve(CHOLMOD_A, L, f_dense, &c);
    memcpy(u_int, u_dense -> x, A -> nrow * sizeof(double));

    inserer_interieur(u_int, u, nb_pt);

    cholmod_free_factor(&L, &c);
    cholmod_free_dense(&f_dense, &c);
    cholmod_free_dense(&u_dense, &c);

    free(f_int);
    free(u_int);
}
```

Équation de Poisson en dimension 2 – Implémentation

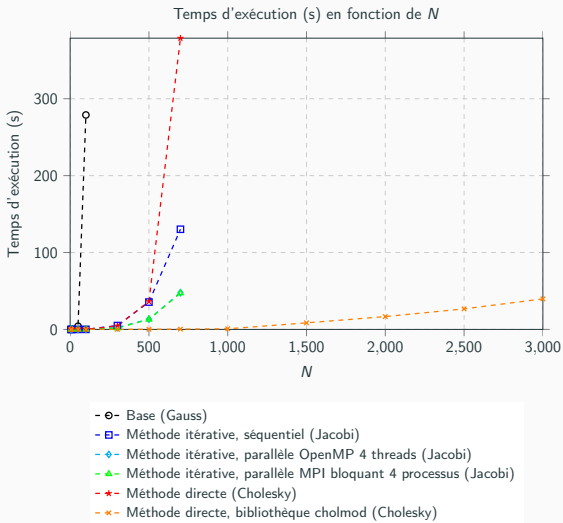
Version avec méthode de résolution directe en séquentiel

Commentaire On note ces résultats :

N	1000	2000	3000	4000	5000
$\ e_h\ _\infty$	0.00000004	0.00000001	<0.00000001	<0.00000001	<0.00000001
Tps d'ex. (s)	0.7	15.7	39.3	79.3	174.5

Équation de Poisson en dimension 2 – Implémentation

Comparaison des performances des méthodes



Équation des ondes en dimension 1

Problème

Soient $L, T, c > 0, D :=]0, L[$ et le problème suivant :

Trouver u de classe C^2 telle que :

$$\left\{ \begin{array}{ll} \frac{\partial^2 u}{\partial t^2}(x, t) - c^2 \frac{\partial^2 u}{\partial x^2}(x, t) = 0 & \forall x \in D, \forall t \in]0, T] \\ u(x, t) = 0 & \forall x \in \partial D, \forall t \in [0, T] \\ u(x, 0) =: u_0(x) & \forall x \in D \\ \frac{\partial u}{\partial t}(x, 0) =: u_1(x) & \forall x \in D \end{array} \right. .$$

Équation des ondes en dimension 1 – Analyse numérique

Schéma numérique

Discrétisation

$$\frac{\partial^2 u}{\partial t^2}(x, t) = \frac{1}{h_t^2} (u(x, t + h_t) - 2u(x, t) + u(x, t - h_t)) + E_{h_t}$$

avec

$$E_{h_t} := -\frac{1}{12} h_t^2 \frac{\partial^4 u}{\partial t^4}(x, t + \theta h_t).$$

Schéma

$$\begin{cases} u_i^1 = h_t u_1(x_i) + u_0(x_i) \\ u_i^{k+1} = -u_i^{k-1} + 2 \left(1 - \frac{c^2 h_t^2}{h^2}\right) u_i^k + \frac{c^2 h_t^2}{h^2} (u_{i-1}^k + u_{i+1}^k) \quad \text{si } k > 0 \end{cases}.$$

Remarques

- Le schéma est explicite.
- Pour $k > 0$, le schéma dépend de valeurs en $k - 1$ et en $k - 2$.

Équation des ondes en dimension 1 – Analyse numérique

Existence et unicité de la solution approchée, consistance du schéma

Proposition Le schéma admet une unique solution.

Proposition Les schémas sont consistants en espace et en temps : $\lim_{h \rightarrow 0} |E_h| = 0$
et $\lim_{h_t \rightarrow 0} |E_{h_t}| = 0$.

Remarque Le schéma est d'ordre 2 pour x et pour y et d'ordre 2 pour t .

Proposition (*admise*) Le schéma est convergent si $c \frac{h_t}{h} \leq 1$.

Remarques

- On vérifiera cette proposition avec un exemple.
- Cette proposition s'appelle la condition de Courant-Friedrichs-Levy (CFL).

Pour la suite, la fonction à approcher sera avec $L = 1$, $T = 1$, $c = 1$,
 $u_0(x) = \sin(\pi x)$ et $u_1(x) = 0$.

Équation des ondes en dimension 1 – Implémentation

Rappel du schéma :

$$\begin{cases} u_i^1 = h_t u_1(x_i) + u_0(x_i) \\ u_i^{k+1} = -u_i^{k-1} + 2 \left(1 - \frac{c^2 h_t^2}{h^2} \right) u_i^k + \frac{c^2 h_t^2}{h^2} (u_{i-1}^k + u_{i+1}^k), \quad \text{si } k > 0 \end{cases}$$

Fonctions qui appliquent le schéma à un nœud (pour $k > 0$ puis pour $k = 0$)

```
static inline __attribute__((always_inline))
double schema(double *u_anc_0, double *u_anc_1, int i, int k){
    double res = // const_1 = pow(c, 2) * pow(h_t, 2) / pow(h, 2)
    -u_anc_1[i]
    + 2 * (1 - const_1) * u_anc_0[i]
    + const_1 * (u_anc_0[i - 1] + u_anc_0[i + 1]);
    return res;
}
```

```
void init_u_1(double (*u_1)(double), double *u_anc_1, double **u_anc_0){
    *u_anc_0 = (double *)malloc(nb_pt * sizeof(double));
    (*u_anc_0)[0] = 0.0; (*u_anc_0)[nb_pt - 1] = 0.0;
    for (int i = 1; i < nb_pt - 1; i++) {
        (*u_anc_0)[i] = h_t * u_1(i * h) + u_anc_1[i];
    }
}
```

Commentaires

- $\|e_{h,h_t}\|_\infty$ en fonction de h et de h_t :

$\downarrow h \quad h_t \rightarrow$	1/100	1/200	1/500	1/1000
1/100	0.01570926	0.00780545	0.00307972	0.00150733
1/200	∞_f	0.00785414	0.00312801	0.00155531
1/500	∞_f	∞_f	0.00314160	0.00156886
1/1000	∞_f	∞_f	∞_f	0.00157080

où ∞_f est ou bien l'infini des flottants double précision (inf), ou bien une valeur très élevée. On vérifie la proposition énoncée, les valeurs ∞_f sont bien atteintes lorsque $c \frac{h_t}{h} > 1$.

- Lorsque h/h_t est fixé, le schéma semble bien converger.
- On ne s'intéresse pas ici aux temps d'exécutions.
- La complexité algorithmique est $O(N \cdot N_t)$.

Équation de la chaleur en dimension 2

Problème

Soient $L, T, a > 0, D :=]0, L[\times]0, L[, f : D \times]0, T] \rightarrow \mathbb{R}$ continue et bornée et le problème suivant :

Trouver u de classe C^2 telle que :

$$\begin{cases} \frac{\partial u}{\partial t}(x, y, t) - a\Delta u(x, y, t) = f(x, y, t) & \forall (x, y) \in D, \forall t \in]0, T] \\ u(x, y, t) = 0 & \forall (x, y) \in \partial D, \forall t \in [0, T] \cdot \\ u(x, y, 0) =: u_0(x, y) & \forall (x, y) \in D \end{cases}$$

Discrétisation

$$\frac{\partial u}{\partial t}(x, y, t) = \frac{1}{h_t} (-u(x, y, t) + u(x, y, t + h_t)) + E_{h_t}$$

avec

$$E_{h_t} := -\frac{1}{2} h_t \frac{\partial^2 u}{\partial t^2}(x, y, t + \theta_t h_t).$$

Schéma

$$u_{i,j}^{k+1} = \alpha u_{i,j}^k + \beta (u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k) + h_t f_{i,j}^k$$

avec

$$\alpha := 1 - \frac{4ah_t}{h^2} \quad \text{et} \quad \beta := \frac{ah_t}{h^2}.$$

Discrétisation

$$\frac{\partial u}{\partial t}(x, y, t + h_t) = \frac{1}{h_t} (u(x, y, t + h_t) - u(x, y, t)) + E_{h_t}$$

avec

$$E_{h_t} := \frac{1}{2} h_t \frac{\partial^2 u}{\partial t^2}(x, y, t + (\theta_t + 1) h_t).$$

Schéma

$$\alpha u_{i,j}^{k+1} + \beta \left(u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^{k+1} + u_{i,j+1}^{k+1} \right) = u_{i,j}^k + h_t f_{i,j}^{k+1}$$

avec

$$\alpha := 1 + \frac{4ah_t}{h^2} \quad \text{et} \quad \beta := -\frac{ah_t}{h^2}.$$

Schéma sous forme matricielle par blocs

$$\boxed{Au^{k+1} = b^k} \Leftrightarrow \underbrace{\begin{pmatrix} \boxed{M} & \boxed{\beta I} & \cdot & \cdot \\ \boxed{\beta I} & \ddots & \ddots & \cdot \\ \cdot & \ddots & \ddots & \boxed{\beta I} \\ \cdot & \cdot & \boxed{\beta I} & \boxed{M} \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_1^{k+1} \\ \vdots \\ \vdots \\ u_{N-1}^{k+1} \end{pmatrix}}_{=:u^{k+1}} = \underbrace{\begin{pmatrix} b_1^{k+1} \\ \vdots \\ \vdots \\ b_{N-1}^{k+1} \end{pmatrix}}_{=:b^{k+1}}$$

avec

$$M := \begin{pmatrix} \alpha & \beta & \cdot & \cdot \\ \beta & \ddots & \ddots & \cdot \\ \cdot & \ddots & \ddots & \beta \\ \cdot & \cdot & \beta & \alpha \end{pmatrix} \quad \text{et} \quad b^k := u^k + h_t f^{k+1}.$$

Discrétisation

$$\frac{1}{2} \left(\frac{\partial u}{\partial t}(x, y, t) + \frac{\partial u}{\partial t}(x, y, t + h_t) \right) = \frac{1}{h_t} (-u(x, y, t) + u(x, y, t + h_t)) + E_{h_t}$$

avec

$$E_{h_t} = O(h_t^2).$$

Schéma

$$\begin{aligned} & \alpha u_{i,j}^{k+1} - \gamma (u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^{k+1} + u_{i,j+1}^{k+1}) \\ &= \beta u_{i,j}^k + \gamma (u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k) + \frac{1}{2} h_t (f_{i,j}^k + f_{i,j}^{k+1}) \end{aligned}$$

avec

$$\alpha := 1 + 4\gamma, \quad \beta := 1 - 4\gamma \quad \text{et} \quad \gamma := \frac{ah_t}{2h^2}.$$

Équation de la chaleur en dimension 2 – Analyse numérique

Schémas numériques – Schéma semi-implicite

Schéma sous forme matricielle par blocs

$$\boxed{Au^{k+1} = b^k} \Leftrightarrow \underbrace{\begin{pmatrix} \boxed{M} & \boxed{-\gamma I} & . & . \\ \boxed{-\gamma I} & \ddots & \ddots & . \\ . & \ddots & \ddots & \boxed{-\gamma I} \\ . & . & \boxed{-\gamma I} & \boxed{M} \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} u_1^{k+1} \\ \vdots \\ \vdots \\ u_{N-1}^{k+1} \end{pmatrix}}_{=:u^{k+1}} = \underbrace{\begin{pmatrix} b_1^{k+1} \\ \vdots \\ \vdots \\ b_{N-1}^{k+1} \end{pmatrix}}_{=:b^{k+1}}$$

avec

$$M := \begin{pmatrix} \alpha & -\gamma & . & . \\ -\gamma & \ddots & \ddots & . \\ . & \ddots & \ddots & -\gamma \\ . & . & -\gamma & \alpha \end{pmatrix}$$

et

$$\forall i, j \in \{1, \dots, N-1\} : b_{i,j}^k := \beta u_{i,j}^k + \gamma (u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k) + \frac{1}{2} h_t (f_{i,j}^k + f_{i,j}^{k+1}).$$

Équation de la chaleur en dimension 2 – Analyse numérique

Existence et unicité des solutions approchées, consistance des schémas

Proposition Les schémas admettent une unique solution.

Proposition Les schémas sont consistants en espace et en temps : $\lim_{h \rightarrow 0} |E_h| = 0$
et $\lim_{h_t \rightarrow 0} |E_{h_t}| = 0$.

Remarque

- Les schémas explicite et implicite sont d'ordre 2 pour x et pour y et d'ordre 1 pour t .
- Le schéma semi-implicite est d'ordre 2 pour x et pour y et d'ordre 2 pour t .

Proposition (*admise*) Le schéma explicite est convergent $\Leftrightarrow \beta \leq \frac{1}{4}$.

Remarques

- On vérifiera cette proposition avec un exemple.
- Cette proposition s'appelle la condition de Courant-Friedrichs-Levy (CFL).

Proposition (*admise*) Les schéma implicite et semi-implicite sont convergents.

Pour la suite, la fonction à approcher sera avec $L = 1$, $T = 1$, $a = 1$, $\lambda = 2a\pi^2$,
 $f(x, y, t) = (-\lambda + 2a\pi^2) \sin(\pi x) \sin(\pi y) e^{-\lambda t}$ et $u_0(x, y) = \sin(\pi x) \sin(\pi y)$.

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en séquentiel

Commentaires

- $\|e_{h,h_t}\|_\infty$ en fonction de h et de h_t :

$\downarrow h \quad h_t \rightarrow$	1/10000	1/20000	1/50000	1/100000
1/10	0.00266777	0.00284805	0.00295614	0.00299216
1/20	0.00039394	0.00057533	0.00068410	0.00072034
1/50	0.00024223	0.00006052	0.00004843	0.00008473
1/100	∞_f	∞_f	0.00004237	0.00000605

On vérifie la proposition énoncée, les valeurs ∞_f sont bien atteintes lorsque $\beta > 1/4$.

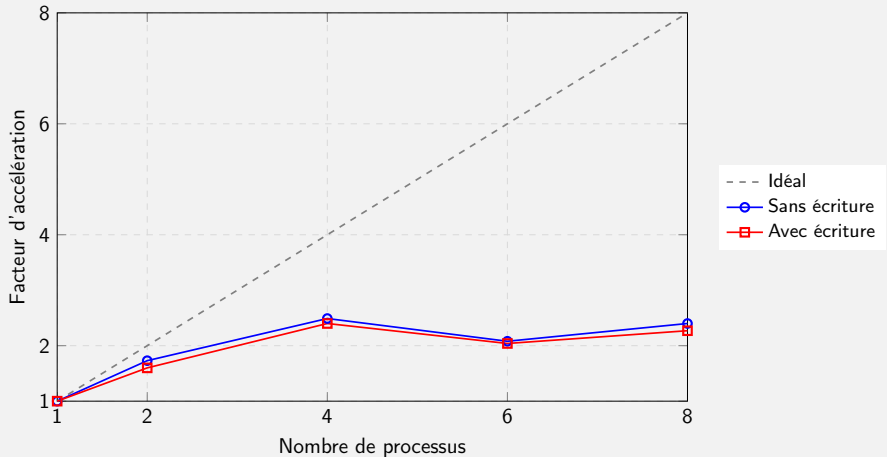
- La condition sur β est très contraignante.
- La complexité algorithmique est $O(N^2 \cdot N_t)$.

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en parallèle avec OpenMP

Commentaire On note ces résultats :

Facteur d'accélération en fonction du nombre de processus

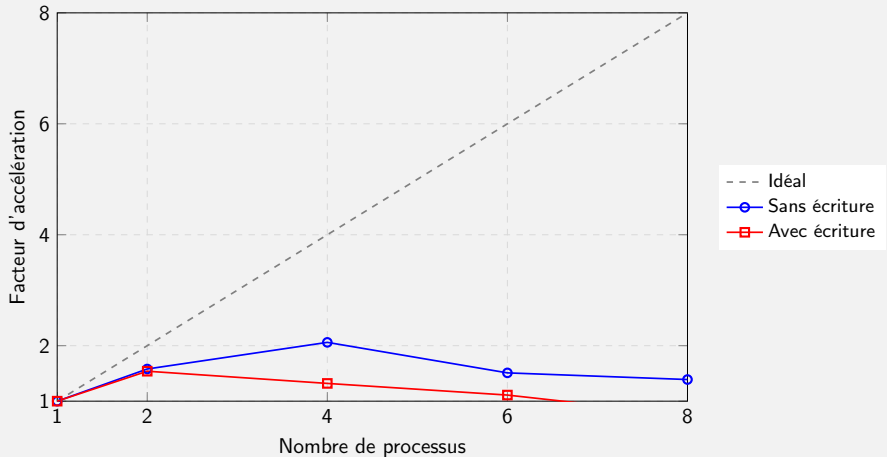


Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma explicite en parallèle avec MPI

Commentaire On note ces résultats :

Facteur d'accélération en fonction du nombre de processus



Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma implicite

Commentaires

- $\|e_{h,h_t}\|_\infty$ en fonction de h et de h_t :

$\downarrow h \quad h_t \rightarrow$	1/10000	1/20000	1/50000	1/100000
1/10	0.00338798	0.00320815	0.00310018	0.00306418
1/20	0.00111862	0.00093767	0.00082903	0.00079281
1/50	0.00048370	0.00030244	0.00019361	0.00015732
1/100	0.00039301	0.00021171	0.00010286	0.00006656

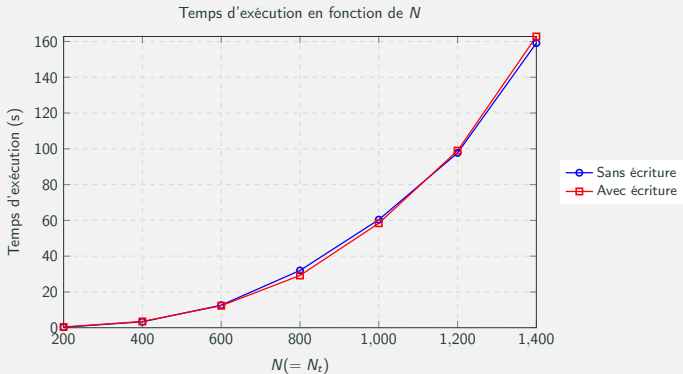
- Les valeurs ∞_f obtenues pour le schéma explicite deviennent proches de 0.

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma implicite

Commentaires

- On note ces résultats :



- La stabilité du schéma implicite est inconditionnelle, ce qui permet d'exécuter sur des pas de temps plus grands.

Équation de la chaleur en dimension 2 – Implémentation

Version avec schéma semi-implicite

Commentaires

- $\|e_{h,h_t}\|_\infty$ en fonction de h et de h_t (avec $L = 1$, $T = 1$, $a = 1$ et $\lambda = 2a\pi^2$) :

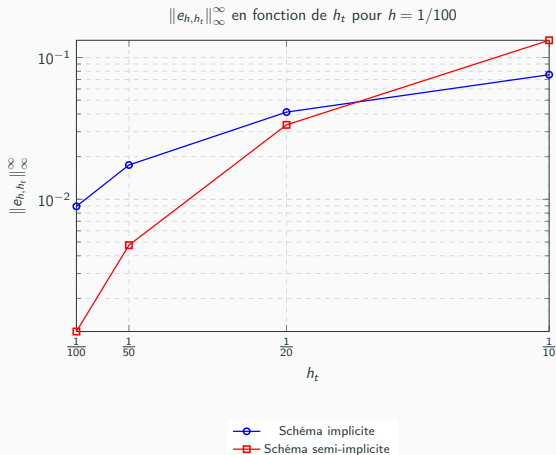
$\downarrow h \quad h_t \rightarrow$	1/10000	1/20000	1/50000	1/100000
1/10	0.00302805	0.00302814	0.00302817	0.00302817
1/20	0.00075646	0.00075655	0.00075657	0.00075658
1/50	0.00012091	0.00012100	0.00012103	0.00012103
1/100	0.00003014	0.00003023	0.00003025	0.00003026

- Les temps d'exécutions sont environ les mêmes que pour le schéma implicite.

Équation de la chaleur en dimension 2 – Implémentation

Comparaison des performances des méthodes

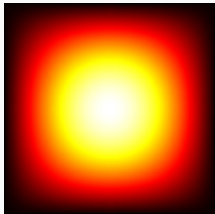
$\|e_{h,h_t}\|_\infty$ en fonction de h (pour $h = 1/100$) et du schéma (implicite et semi-implicite) :



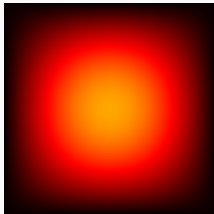
Équation de la chaleur en dimension 2 – Visualisation

On peut visualiser la simulation avec Python et matplotlib.

Diffusion de la chaleur



État initial :
 $t = 0.0$



État après
diffusion : $t = 0.1$



État après
diffusion : $t = 0.2$



État après
diffusion : $t = 0.3$

Bibliographie et supports

- *Rapports de calcul scientifique*. (2008) par Patrick Ciarlet
- *Finite-Difference Approximations to the Heat Equation* (2004) par Gerald W. Recktenwald
- *Numerical Methods for Ordinary Differential Equations* par Habib Ammari et Konstantinos Alexopoulos
- *Lecture 6: Finite difference methods* par Habib Ammari
- *SUITESPARSE : A SUITE OF SPARSE MATRIX SOFTWARE*
- *Direct Methods for Sparse Linear Systems* par Timothy A. Davis
- Cours de calcul numérique (M1 CHPS) par Serge Dumont
- Cours d'analyse et calcul numérique (L3 Maths) par Francesco Bonaldi
- Cours d'algorithmique et programmation parallèle (M1 CHPS) par David Defour
- Forums d'aides

Lien vers le dépôt récent GitHub du projet

<https://github.com/gaillot18/PDE-FDM-HPC>

(contient le rapport écrit, la présentation et le projet)

Lien vers le dépôt initial GitHub du projet

<https://github.com/gaillot18/Stage-M1-CHPS.git>

Merci pour votre attention.