Calcul Hautes Performances

Semestre 8 – EM8AN202 – M2 Groupe 1

Projet Calcul Hautes Performances

Rendu à :
Pr. H. Beaugendre

Rendu par :
Antoine Boucher
Gabriel Rodiere

Le Mercredi 24 Mai 2023.

I Analyse du problème

Nous nous plaçons dans le domaine $\Omega = [0\,;\mathrm{L}_x] \times [0\,;\mathrm{L}_y]$ de $\mathbf{R}^2$ dans lequel nous résolvons l'équation de la chaleur $(\mathcal{E})$:

$$(\mathcal{E}) \iff \begin{cases} \partial_t u - \mathrm{D}\,\Delta u = f & \text{sur } \Omega \setminus \partial\Omega \\ u = g & \text{sur } \Gamma_0 \\ u = h & \text{sur } \Gamma_1 \end{cases}$$

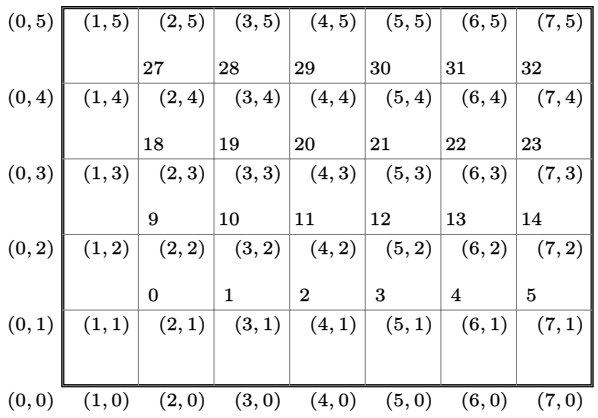

FIGURE 1 – Domaine Ω où nous réalisons le maillage, avec $\mathrm{N}_x = 6$ et $\mathrm{N}_y = 4$.

(a) Nous procédons au maillage du domaine rectangle Ω. Posons :

$$\begin{cases} x_i = i\,\Delta x & \text{avec} \quad \Delta x = \mathrm{L}_x/(\mathrm{N}_x + 1) \\ y_j = j\,\Delta y & \text{avec} \quad \Delta y = \mathrm{L}_y/(\mathrm{N}_y + 1) \end{cases}$$

avec i et j qui prennent leurs valeurs dans $[\![0\,;\mathrm{N}_x + 1]\!] \times [\![0\,;\mathrm{N}_y + 1]\!]$.
Puis, nous discrétisons la dimension du temps en posant $t^n = n\,\Delta t$.
Enfin, en notant $u_{i,j}^n$ (resp. $f_{i,j}^n$) l'approximation des valeurs exactes $u(x_i, y_j, t^n)$ (resp. $f(x_i, y_j, t^n)$) faite en négligeant les $\mathcal{O}(\Delta x^2)$, $\mathcal{O}(\Delta y^2)$ et $\mathcal{O}(\Delta t)$, il vient :

$$\begin{aligned} u_{i,j}^{n+1} = u_{i,j}^n + \Delta t\, f_{i,j}^{n+1} + \mathrm{D}\frac{\Delta t}{\Delta x^2}\Big(u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}\Big) \\ + \mathrm{D}\frac{\Delta t}{\Delta y^2}\Big(u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}\Big). \end{aligned}$$

En réarrangeant, nous obtenons :

$$\begin{aligned} -\mathrm{D}\frac{\Delta t}{\Delta x^2}\Big(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}\Big) + \left[1 + 2\mathrm{D}\Delta t\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\right] u_{i,j}^{n+1} \\ - \mathrm{D}\frac{\Delta t}{\Delta y^2}\Big(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\Big) = u_{i,j}^n + \Delta t\, f_{i,j}^{n+1}. \end{aligned}$$

Ces équations tiennent pour $(i,j) \in [\![1\,;\mathrm{N}_x]\!] \times [\![1\,;\mathrm{N}_y]\!]$. Pour les termes aux bords $u_{i,j}^{n+1}$, cela fait apparaître au second membre un, voire deux, terme(s) supplémentaire(s) $g_{i,j}^{n+1}$ et $h_{i,j}^{n+1}$ à la place.

(b) Introduisons un indice permettant de mettre $u_{i,j}^n$ sous forme d'un vecteur U^n de taille $\mathrm{N}_x \times \mathrm{N}_y$. En utilisant la notation pour le C++ (figure 1), nous le définissons de la façon suivante :

$$\mathrm{J} = (j-1)\mathrm{N}_x + (i-1),$$

qui a une dépendance linéaire en i. Alors, il nous vient :

$$\begin{cases} i = 1 + (\mathrm{J} \mod \mathrm{N}_x) \\ j = 1 + \lfloor \mathrm{J}/\mathrm{N}_x \rfloor \end{cases}$$

Et ainsi nous avons $u_{\mathrm{J}} = u_{i,j}$.
Nous pouvons alors réécrire le système sous forme matricielle :

$$(\boldsymbol{\mathcal{I}}_{\mathrm{N}_x \times \mathrm{N}_y} + \mathrm{D}\Delta t\, \boldsymbol{\mathcal{L}})\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \left(\mathbf{F}^{n+1} + \mathbf{G}^{n+1} + \mathbf{H}^{n+1}\right) = \mathbf{K}^n$$

où $\boldsymbol{\mathcal{I}}_{\mathrm{N}_x \times \mathrm{N}_y}$ est la matrice identité de taille $(\mathrm{N}_x \times \mathrm{N}_y)^2$ et $\boldsymbol{\mathcal{L}}$ est la matrice du Laplacien en 2 dimensions et avec les vecteurs de conditions aux limites :

$$\mathbf{G}^{n+1} = \frac{\mathrm{D}}{\Delta y^2} \begin{pmatrix} \left[g_{i,0}^{n+1}\right]_{\mathrm{N}_x}^1 \\ 0 \\ \vdots \\ 0 \\ \left[g_{i,\mathrm{N}_y+1}^{n+1}\right]_{\mathrm{N}_x}^1 \end{pmatrix} \quad \text{et} \quad \mathbf{H}^{n+1} = \frac{\mathrm{D}}{\Delta x^2} \begin{pmatrix} \begin{bmatrix} h_{0,j}^{n+1} \\ 0 \\ \vdots \\ 0 \\ h_{\mathrm{N}_x+1,j}^{n+1} \end{bmatrix}_{\mathrm{N}_y}^1 \end{pmatrix}$$

dans $\mathcal{M}_{1,\mathrm{N}_x \times \mathrm{N}_y}$.

(c) La matrice du Laplacien s'écrit :

$$\boldsymbol{\mathcal{L}} = \begin{pmatrix} \boldsymbol{\mathcal{X}} & \boldsymbol{\mathcal{Y}} & & \\ \boldsymbol{\mathcal{Y}} & \ddots & \ddots & \\ & \ddots & \ddots & \boldsymbol{\mathcal{Y}} \\ & & \boldsymbol{\mathcal{Y}} & \boldsymbol{\mathcal{X}} \end{pmatrix}$$

avec les matrices $\boldsymbol{\mathcal{X}}$ et $\boldsymbol{\mathcal{Y}}$ de taille $\mathrm{N}_x \times \mathrm{N}_x$ définies par

$$\boldsymbol{\mathcal{X}} = \mathrm{tridiag}\left(-\frac{1}{\Delta x^2}, \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}, -\frac{1}{\Delta x^2}\right) \quad \text{et} \quad \boldsymbol{\mathcal{Y}} = -\frac{1}{\Delta y^2}\boldsymbol{\mathcal{I}}_{\mathrm{N}_x}$$

Nous en déduisons aisément que $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{I}}_{\mathrm{N}_x \times \mathrm{N}_y} + \mathrm{D}\Delta t\boldsymbol{\mathcal{L}}$ est la matrice à inverser par la méthode du gradient conjugué.

(d) La matrice $\boldsymbol{\mathcal{A}}$ est :

— symétrique réelle (par construction),

— pentadiagonale (par construction)

Ainsi, d'après le théorème spectral, les valeurs propres de $\boldsymbol{\mathcal{A}}$ sont réelles. De plus, d'après le théorème des disques de Gerschgorin, les valeurs propres $\{\lambda_i\}$ de $\boldsymbol{\mathcal{A}}$ sont supérieures ou égales à 1 car $a_{i,i} > 0$ et $\boldsymbol{\mathcal{A}}$ est à diagonale strictement dominante :

$$a_{i,i} = 1 + \mathrm{D}\Delta t \left[2/\Delta x^2 + 2/\Delta y^2\right] \geqslant 1 + \sum_{\substack{j=1 \\ j \neq i}}^{\mathrm{N}_x \times \mathrm{N}_y} |a_{i,j}| > \sum_{\substack{j=1 \\ j \neq i}}^{\mathrm{N}_x \times \mathrm{N}_y} |a_{i,j}|$$

La matrice $\boldsymbol{\mathcal{A}}$ est donc :

— définie-positive,

— inversible.

Par conséquent, le système admet une unique solution. On notera également que $\boldsymbol{\mathcal{A}}$ est une M-matrice car ses coefficients extra-diagonaux sont en outre tous négatifs ou nuls (Z-matrice), ce qui garantit que la solution par différences finies jouit de la même propriété de monotonie que la solution exacte $u(x, y, t)$ (autrement dit $\mathbf{U}^n$ est positive dès que $\mathbf{K}^0$ l'est). C'est le *principe du maximum discret*, c'est une vérification supplémentaire que le problème est correctement approché par le solveur du Gradient Conjugué.

II Code séquentiel

Il faut regarder le fichier README.txt dans le dossier sequentiel pour avoir des instructions sur la compilation.

II.1 Structure du code

Nous cherchons donc à résoudre le problème de diffusion posé, avec conditions aux bords, en séquentiel dans un premier temps. Les 3 cas tests suivant seront utilisés :

Cas n°1 (solution stationnaire)

$$f(x,y) = 2(y - y^2 + x - x^2), \quad g = 0, \quad h = 0$$

Cas n°2 (solution stationnaire)

$$f(x,y) = g(x,y) = h(x,y) = \sin(x) + \cos(y)$$

Cas n°3 (solution instationnaire)

$$f(x,y,t) = \exp\left[-\left(x - \frac{\mathrm{L}_x}{2}\right)^2 - \left(y - \frac{\mathrm{L}_y}{2}\right)^2\right] \cos\left(\frac{\pi}{2}t\right), \quad g = 0, \quad h = 1$$

Notre code contient quatre fichier .cpp.

1. fonctions.cpp, ce fichier contient :
   - la lecture des paramètres `Nx, Ny, Lx, Ly, D, dt, tmax, testCase` dans le fichier parameters.txt, où `testCase` permet de sélectionner les fonctions f, g et h selon le cas choisi.
   - la définition des fonction f, g et h.
   - une fonction `saveSolution(...)` qui sauvegarde la solution dans un fichier texte.
   - une fonction qui crée un script Gnuplot et permet d'afficher directement la visualisation des résultats dans Gnuplot. (*NB : sur Mac l'affichage est automatique, sur les ordinateurs de l'école il faut récupérer le fichier .gif dans le fichier solution*)
2. conjGrad.cpp, ce fichier contient :
   - la fonction `conjugateGradient(...)` qui permet de résoudre le système par la méthode du gradient conjugué.
   - les fonctions associées permettant de faire un produit scalaire.
3. prodMatVect_BC.cpp, ce fichier contient :
   - la fonction `prodMatvect(...)` qui construit le produit matrice vecteur AU.
   - la fonction `calculateRightHandSide(...)` qui de la même manière construit le second membre du système.
4. main.cpp, ce fichier contient :
   - la fonction `main(...)` qui effectue la boucle en temps correspondant au schéma d'Euler implicite.
   - la fonction `calculateSolutionStep(...)`.

II.2 Validation du code

Voici l'allure des solutions obtenues pour les 3 cas test avec $\mathrm{N}x = \mathrm{N}y = 100$.

II.2.1 Affichage de la solution

Nous pouvons afficher les solutions obtenues par notre code :

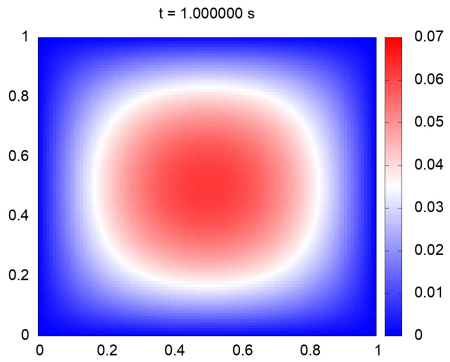

FIGURE 2 – Solution numérique du cas 1 à $t_{\max} = 1s$

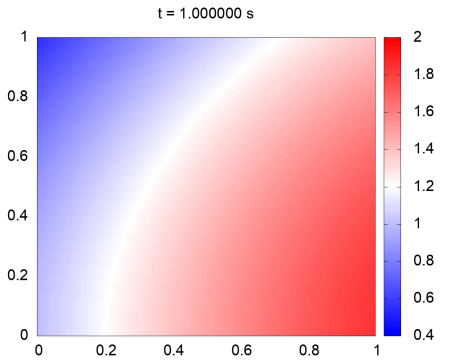

FIGURE 3 – Solution numérique du cas 2 à $t_{\max} = 1s$

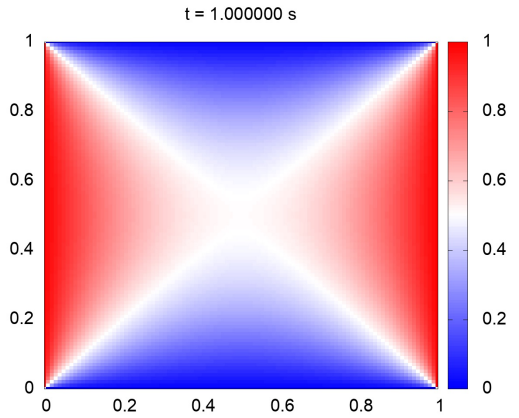

FIGURE 4 – Solution numérique du cas 3 à $t_{\max} = 1s$

Comme on peut voir sur les figures 2, 3 et 4, les résultats obtenus sont similaires aux solutions de référence donc nous pouvons valider notre code.

II.2.2 Calculs d'erreurs

Nous calculons l'erreur en norme L^2 d'approximation :

$$\|e\|_{\mathrm{L}^2} = \sqrt{\max(dx, dy)} * \|\mathrm{U} - \mathrm{U}_{\mathrm{ex}}\|_2$$

où U_{ex} est la solution exacte.

Dans le cas 1,

$$\mathrm{U}_{\mathrm{ex}}(x, y) = x(1-x)y(1-y)$$

et dans le cas 2,

$$\mathrm{U}_{\mathrm{ex}}(x, y) = \cos x + \sin y.$$

Il n'y a pas de solution analytique simple au cas 3, nous vérifierons notre donc notre code sur les deux premiers cas.

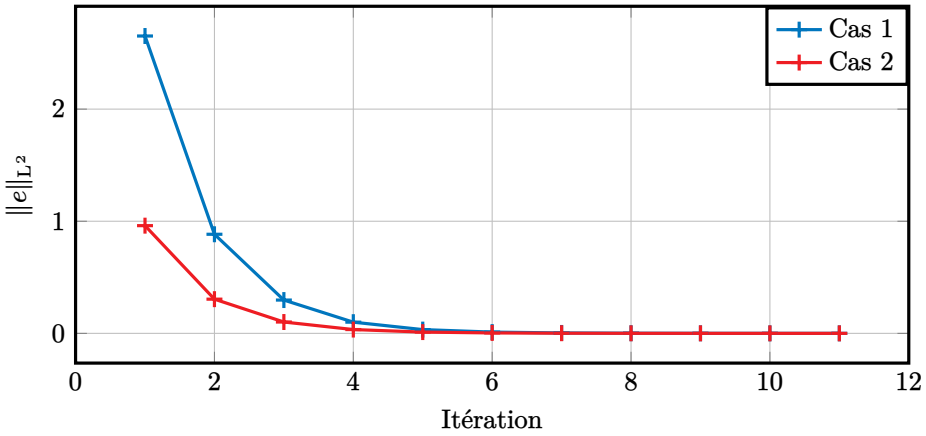

FIGURE 5 – Erreur en norme L^2 selon les itérations – Code séquentiel

Nous voyons que l'erreur peut être considérée nulle pour un nombre d'itérations supérieur à 6. Donc le code séquentiel converge bien et il est vérifié.

III Code parallèle

Il faut regarder le fichier README.txt dans le dossier parallele pour avoir des instructions sur la compilation.

Dans cette partie, les fonctions utilisées restent les mêmes. Nous avons parallélisé le gradient conjugué, la fonction réalisant le produit matrice vecteur AU ainsi que la fonction construisant le second membre. Nous stockons les informations relatives au processeur `me` dans une variable `procDF` (pour proc Datafile) de type `procData` (iBeg, iEnd, n, `me`, `locSize`, `tag`, `status`).

III.1 Répartition des inconnues

Les inconnues du problème sont réparties entre plusieurs processeurs dont le nombre total est noté n_{proc}. Pour faire fonctionner notre code correctement, nous devons imposer la condition : $n_{\text{proc}} \leq \text{N}y$. La fonction charge permet de répartir les calculs entre les différents processeurs et permet de les partager de la façon la plus équilibrée possible sur des portions $[\![\text{iBeg}_{\text{me}} ; \text{iEnd}_{\text{me}}]\!]$. Ainsi, chaque processeur ne connaît qu'une petite partie des vecteurs utilisés.

III.2 Fonction du gradient conjugué

Cette fonction reste quasiment la même que dans la version séquentielle. Nous utilisons simplement des `MPI_All_Reduce()` *in place* pour calculer les normes des restes. Chaque processus réalise une partie de la résolution.

III.3 Produit Matvect et second membre

Le second membre est calculé avec un nombre réduit de composantes, en prenant en compte le décalage des indices.

Pour le produit matrice vecteur en revanche, il nécessite de connaître $2\text{N}x + $ `locSize` inconnues. Il faut donc faire des communications pour partager les coordonnées manquantes afin de compléter le produit.

Nous stockons le vecteur augmenté des communications dans une variable `wide_Uloc` :

$$\texttt{wide\_Uloc} = \begin{bmatrix} \texttt{U2} \\ \texttt{U} \\ \texttt{U1} \end{bmatrix}.$$

où `U1` et `U2` sont respectivement les Nx premiers termes de me+1 et les Nx derniers termes de me−1.

Bien sûr pour `me` $= 0$ ou $n_{\text{proc}} - 1$, il ne faut communiquer que la partie de me+1 ou de me−1 respectivement. De façon générale, les communications s'écrivent ainsi :

```
// On envoie les Nx premiers termes à me-1
 MPI_Send(&U[0], Nx, MPI_DOUBLE, procDF.me - 1, procDF.tag, MPI_COMM_WORLD);
 // On envoie les Nx derniers termes à me+1
 MPI_Send(&U[procDF.locSize - Nx], Nx, MPI_DOUBLE, procDF.me + 1, procDF.tag,
↪  MPI_COMM_WORLD);

 // On reçoit les Nx premiers termes de me+1
 MPI_Recv(&U1[0], Nx, MPI_DOUBLE, procDF.me + 1, procDF.tag, MPI_COMM_WORLD,
↪  MPI_STATUS_IGNORE);
 // On reçoit les Nx derniers termes de me-1
 MPI_Recv(&U2[0], Nx, MPI_DOUBLE, procDF.me - 1, procDF.tag, MPI_COMM_WORLD,
↪  MPI_STATUS_IGNORE);
```

avec une inversion des envois et reçus selon que le processeur est pair ou impair, ceci afin d'éviter la saturation.

III.4 Fonction Main

La fonction `main()` réalise les mêmes actions qu'en séquentiel, sauf qu'elle concatène la solution `U` sur le processeur racine 0 via la `concatenateSolutionToRootProcessAndSave()`.

Le calcul du temps de calcul est fait localement puis globalement en prenant le max des temps de calcul sur tous les processeurs.

III.5 Validation du code

Nous traçons l'erreur en norme L^2 pour chaque cas test et nous retrouvons le même comportement que le lors de l'utilisation du code séquentiel, ce qui valide notre code parallèle.

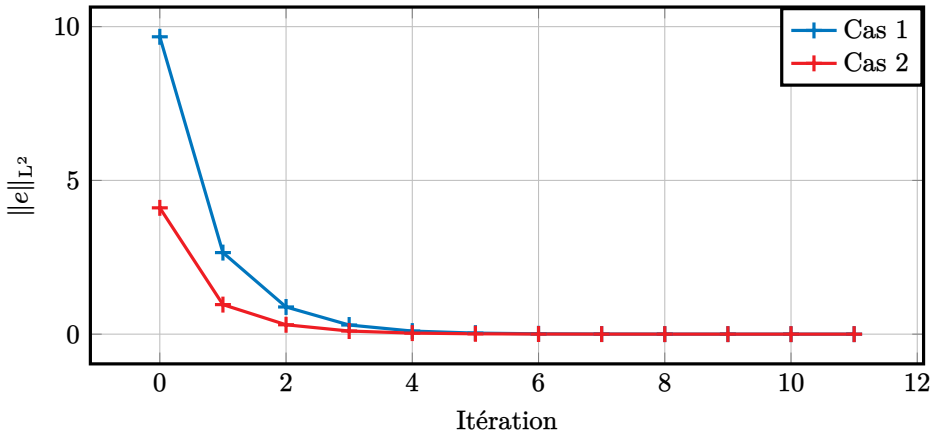

FIGURE 6 – Erreur en norme L^2 selon les itérations – Code parallèle

IV Comparaison entre le code séquentiel et parallèle

Comme nous avons pu le voir, les résultats sont identiques pour les méthodes séquentielles et parallèles. Il est donc intéressant de tracer les courbes de speed-up et d'efficacité.

IV.1 Speed-up

Le speed-up est définit par le rapport du temps de calcul séquentiel et du temps de calcul parallèle tel que :

$$S = \frac{T_{\text{séquentiel}}}{T_{\text{parallèle}}}$$

On ne compte bien sûr pas le temps d'écriture dans les fichiers.

On a comme référence : $T_{\text{séquentiel}} = 0{,}281871$ s. Nous faisons les calculs à la main et nous traçons le speed-up suivant sur le cas test 3 pour $n_{\text{proc}} = 8$:

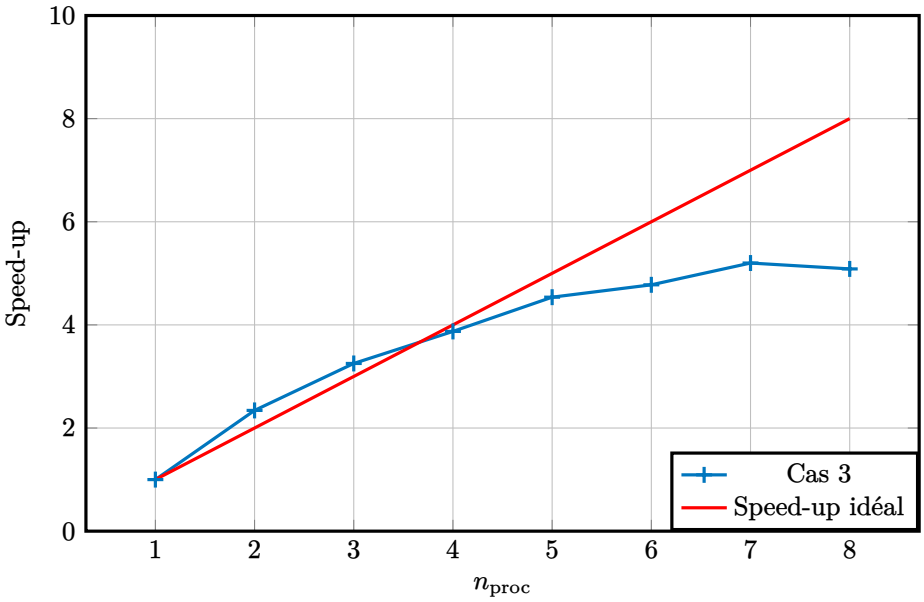

FIGURE 7 – Speed-up S du code parallèle en fonction du nombre n_{proc} de processeur activés.

IV.2 Efficacité

L'efficacité se définit elle par le rapport entre le speed-up et le nombre de processeur : $e = \frac{S}{n_{\text{proc}}}$

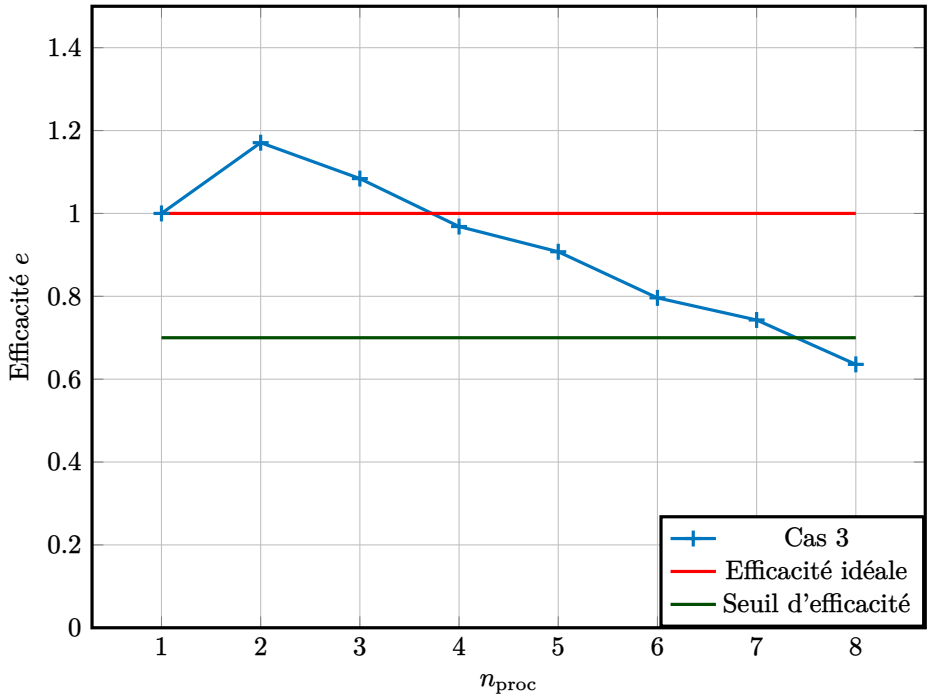

FIGURE 8 – Efficacité e du code parallèle en fonction du nombre de processeur activés n_{proc}

Le speed-up est considéré parfait lorsque $S = n_{\text{proc}}$ d'où le fait que l'efficacité soit considérée parfaite lorsque $e = 1$. Sur la figure 7, on constate que le speed-up n'augmente plus au bout d'un certain nombre de processeur $n_{\text{proc}} = 7$. Sur la figure 8, on constate également que l'efficacité diminue lorsque le nombre de processeur augmente, *i.e.* lorsque $n_{\text{proc}} \geqslant 4$ (on considère par rapport à l'efficacité idéale).

Ceci s'explique par le nombre de communications croissant : lorsque le nombre de processeur augmente, le nombre des communication augmente et le coût en temps des communications devient alors plus important que le coup de calcul économisé.

V Conclusion

Ce projet d'initiation au calcul parallèle a démontré son importance dans la résolution numérique de l'équation de diffusion. Après avoir discrétisé le domaine de résolution et reformulé le système d'équations aux différents points du maillage, nous l'avons représenté sous forme matricielle. Initialement, la résolution de ce système a été implémentée de manière séquentielle grâce à un solveur de type Gradient Conjugué, car il est essentiel de valider cette étape avant de paralléliser.

La parallélisation des fonctions calculant le produit matrice-vecteur, le second membre et le gradient conjugué a permis de paralléliser quasi-complètement le code de résolution de l'équation de diffusion. Une répartition adéquate des inconnues sur les n_{proc} processeurs et l'établissement des communication nécessaires entre ces derniers ont été essentiels pour optimiser le code parallèle.

Les erreurs quadratiques ont été utilisées pour valider nos codes séquentiels et parallèles. Une fois le code validé, il était également important d'évaluer l'apport du calcul parallèle. Le speed-up et l'efficacité ont été utilisés comme outils pour quantifier les gains du calcul parallèle. En effet, la parallélisation du code visait à réduire le temps de calcul en répartissant la charge de travail entre les différents processeurs. Cependant, si le nombre de processeurs est trop élevé, le coup de calcul peut être amené à augmenter (comme démontré en IV.2). Il est donc crucial de trouver le nombre optimal de processeurs pour obtenir les meilleures performances possible.

Pour notre machine, la parallélisation est idéale avec 7 processeurs.