Travaux dirigés n°1

Xavier JUVIGNY

January 13, 2024

Contents

1 Produit matrice–matrice **1**

2 Parallélisation MPI **2**
2.1 Circulation d'un jeton dans un anneau 2
2.2 Calcul très approché de pi 2
2.3 Diffusion d'un entier dans un réseau hypercube* 3

1 Produit matrice–matrice

Soient A et B deux matrices définies à l'aide de deux couples de vecteurs $\{u_A, v_A\}$ et $\{u_B, v_B\}$:

$$\left\{ \begin{array}{rcl} A & = & u_A.v_A^T \text{ soit } A_{ij} = u_{A_i}.v_{A_j} \\ B & = & u_B.v_B^T \text{ soit } B_{ij} = u_{B_i}.v_{B_j} \end{array} \right. .$$

On calcule le produit matrice–vecteur C=A.B à l'aide d'un produit matrice–matrice plein (complexité de $2.n^3$ opérations arithmétiques) et on valide le résultat obtenu à l'aide de l'expression sous forme de produit tensoriel de A et B :

$$\begin{array}{rcccccl} C & = & A.B & = & \left(u_A.v_A^T\right).\left(u_B.v_B^T\right) & = & u_A\left(v_A^T.u_B\right)v_B^T \\ & = & u_A\left(v_A|u_B\right)v_B^T & = & \left(v_A|u_B\right)u_A.v_B^T & & \end{array} .$$

Soit :

$$C_{ij} = \left(v_A|u_B\right)u_{A_i}.v_{B_j},$$

ce qui nécessite en tout $2.n + 2.n^2$ opérations arithmétiques (dont $2.n$ opérations pour le produit scalaire).

On se propose, par étape, de paralléliser en mémoire partagée le produit matrice–matrice fourni dans le fichier `ProdMatMat.cpp`. L'exécutable pour tester le produit matrice-matrice est `TestProduitMatrix.exe` :

1. Mesurez le temps de calcul du produit matrice–matrice donné en donnant en entrée diverses dimensions. Essayez en particulier de prendre pour dimension 1023, 1024 et 1025 (il suffit de passer la dimension en argument à l'exécution. Par exemple `./TestProductMatrix.exe 1023` testera le produit matrice-matrice pour des matrices de dimension 1023). En vous servant des transparents du cours, expliquez clairement les temps obtenus.

2. **Première optimisation :** Permutez les boucles en i, j et k jusqu'à obtenir un temps optimum pour le calcul du produit matrice–matrice (et après vous être persuadé que cela ne changera rien au résulat du calcul). Expliquez pourquoi la permutation des boucles optimale que vous avez trouvée est bien la façon optimale d'ordonner les boucles en vous servant toujours du support de cours.

3. **Première parallélisation :** A l'aide d'OpenMP, parallélisez le produit matrice–matrice. Mesurez le temps obtenu en variant le nombre de threads à l'aide de la variable d'environnement `OMP_NUM_THREADS`. Calculez l'accélération et le résultat obtenu en fonction du nombre de threads, commentez et expliquez clairement ces résultats.

4. Argumentez et donnez clairement la raison pour laquelle il est sûrement possible d'améliorer le résultat que vous avez obtenu.

5. **Deuxième optimisation :** Pour pouvoir exploiter au mieux la mémoire cache, on se propose de transformer notre produit matrice–matrice "scalaire" en produit matrice–matrice par bloc (on se servira pour le produit "bloc–bloc" de la meilleure version **séquentielle** du produit matrice–matrice obtenu précédemment).

   L'idée est de décomposer les matrices A, B et C en sous-blocs matriciels :

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ A_{N1} & & & A_{NN} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} & \dots & B_{1N} \\ B_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ B_{N1} & & & B_{NN} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} & \dots & C_{1N} \\ C_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ C_{N1} & & & C_{NN} \end{pmatrix},$$

   où A_{IJ}, B_{IJ} et C_{IJ} sont des sous–blocs possédant une taille fixée (par le programmeur).

   Le produit matrice–matrice se fait alors par bloc. Pour calculer le bloc C_{IJ}, on calcule :

$$C_{IJ} = \sum_{K=1}^{N} A_{IK}.B_{KJ}.$$

   Mettre en œuvre ce produit matrice–matrice en séquentiel puis faire varier la taille des blocs jusqu'à obtenir un optimum.

6. Comparer le temps pris par rapport au produit matrice–matrice "scalaire". Comment interprétez vous le résultat obtenu ?

7. **Parallélisation du produit matrice–matrice par bloc** : À l'aide d'OpenMP, parallélisez le produit matrice–matrice par bloc puis mesurez l'accélération parallèle en fonction du nombre de threads. Comparez avec la version scalaire parallélisée. Comment expliquez vous ce résultat ?

8. **Comparaison avec blas**: Comparez vos résultat avec l'exécutable `produit_matmat_blas.exe` qui utilise un produit matrice-matrice optimisé. Quel rapport de temps obtenez vous ? Quelle version est la meilleure selon vous ?

2 Parallélisation MPI

Ecrivez en langage Python les programmes suivants.

2.1 Circulation d'un jeton dans un anneau

Ecrivez un programme tel que :

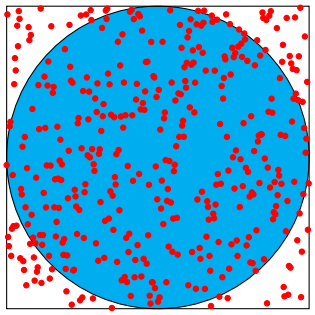
1. le processus de rang zéro initialise un jeton à 1 puis l'envoie au processus de rang un;
2. le processus de rang un reçoit le jeton, l'incrémente de un puis l'envoie au processus de rang deux;
3. ...
4. le processus de rang p reçoit le jeton, l'incrémente de un puis l'envoie au processus de rang $p+1$ $(0 < p < nbp - 1)$;
5. ...
6. le processus de rang nbp $-$ 1 reçoit le jeton, l'incrémente de un et l'envoie au processus de rang zéro;
7. le processus de rang zéro reçoit le jeton du processus nbp $-$ 1 et l'affiche à l'écran.

2.2 Calcul très approché de pi

On veut calculer la valeur de pi à l'aide de l'algorithme stochastique suivant :

- on considère le carré unité $[-1;1] \times [-1;1]$ dans lequel on inscrit le cercle unité de centre $(0,0)$ et de rayon 1;
- on génère des points aléatoirement dans le carré unité;
- on compte le nombre de points générés dans le carré qui sont aussi dans le cercle;
- soit r ce nombre de points dans le cercle divisé par le nombre de points total dans le carré, on calcule alors pi comme $\pi = 4.r$.

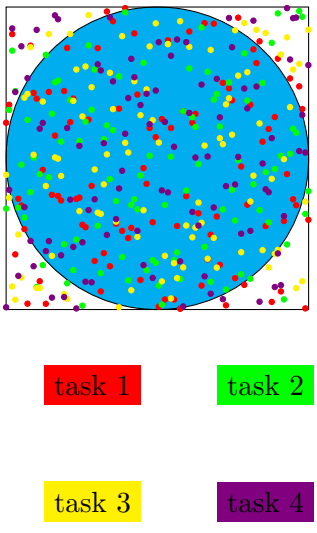
Remarquez que l'erreur faite sur pi décroit quand le nombre de points générés augmente.



Couper l'itération de boucle en plusieurs morceaux pouvant être exécutés par différentes tâches simultanément:

- chaque tâche exécute sa portion de boucle;
- chaque tâche peut exécuter son travail sans avoir besoin d'information des autres tâches (indépendance des données);
- la tâche maître (que le programmeur aura choisi parmi ses tâches) reçoit le résultat des autres tâches à l'aide d'échanges de message point à point;
- utiliser l'échange de message global adéquat pour obtenir le résultat final.

Mesurez le temps mis par les deux versions (séquentiel et parallèle) et calculez l'accélération obtenue. Est-ce cohérent avec votre machine (cf. `lscpu` ou gestionnaire de tâches) ?

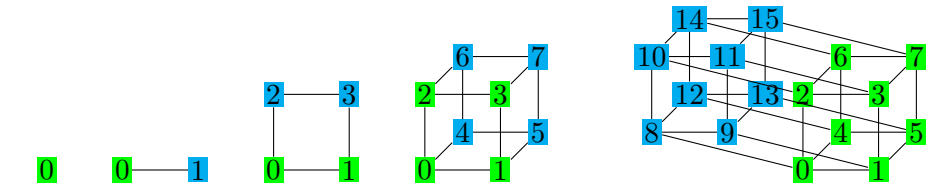


Le concept : diviser le travail parmi les tâches disponibles en communicant des données à l'aide d'appel à des fonctions d'envoi/réception point à point.

2.3 Diffusion d'un entier dans un réseau hypercube*

On veut écrire un programme qui diffuse un entier dans un réseau de nœuds de calculs dont la topologie est équivalente à celle d'un hypercube de dimension d (et qui contient donc 2^d nœuds de calcul).

1. Écrire un programme qui diffuse un entier dans un hyper cube de dimension 1 :
   - la tâche 0 initialise un jeton à une valeur entière choisie par le programmeur et envoie cette valeur à la tâche 1;
   - la tâche 1 reçoit la valeur du jeton de la tâche 0;
   - les deux tâches affichent la valeur du jeton.
2. Diffuser le jeton généré par la tâche 0 dans un hypercube de dimension 2 de manière que cette diffusion se fasse en un minimum d'étapes (et donc un maximum de communications simultanées entre tâches).
3. Faire de même pour un hypercube de dimension 3.
4. Écrire le cas général quand le cube est de dimension d. Le nombre d'étapes pour diffuser le jeton devra être égal à la dimension de l'hypercube.



Hypercubes de dimension 0, 1, 2, 3 et 4