Algorithmique et programmation parallèle : DM

William AUFORT et Raphaël CHARRONDIÈRE

7 Décembre 2014

1 Automate cellulaire

Note : Dans toute cette partie, on note p^2 le nombre de processeurs, N la taille de la grille (qui contient donc N^2 éléments). On supposera comme à l'habitude que p divise N.

Question 1

Pour calculer X^{t+1} à partir de X^t il faut appliquer la fonction δ à tous les éléments de X^t, soit N^2 fois. Donc, pour calculer X^t à partir de X^0, il faut répéter cette opération t fois, ce qui donne tN^2 applications de la fonction δ nécessaires.

Question 2

Nous donnons ici l'implémentation de l'automate cellulaire de la partie 3, et non pas l'implémentation de l'automate général (où les neuf voisins sont nécessaires pour la mise à jour). Nous expliquerons les différences après la description de l'algorithme.

L'idée est de disposer les processeurs selon une grille. Chaque processeur dispose d'un bloc de $(cacNp)^2$ éléments.

Considérons un processeur P. Celui-ci dispose des données $(\widehat{x_{i,j}})$. Pour effectuer un calcul $\delta(\widehat{x_{i,j}})$, P a besoin en plus de $\widehat{x_{i-1,j}}$, $\widehat{x_{i+1,j}}$, $\widehat{x_{i,j-1}}$ et $\widehat{x_{i,j+1}}$. On peut calculer l'image des éléments de P_i qui ne sont pas à la frontière (les éléments en rouge sur la figure 1), P_i n'a besoin que de valeurs qui lui sont connues. Les éléments à la frontière nécessitent quant à eux une ou deux valeurs dont disposent les processeurs voisins de P_i sur la grille (les éléments en bleu sur la figure 1) pour pouvoir être mis à jour.

Chaque processeurs va donc recevoir deux lignes (provenant de ses voisins horizontaux) et deux colonnes (provenant de ses voisins verticaux). Chaque processeurs doit donc également envoyer les lignes (respectivement les colonnes) supérieures et inférieures dont ont besoin les processeurs voisins horizontaux (repectivement verticaux), qui correspondent exactement aux éléments de la frontière.

Il est intéressant de remarquer que ces envois et réceptions peuvent être fait en parallèle, mais également pendant les calculs pour les éléments qui ne sont pas sur la frontière.

Cas général Dans le cas général, il y a une petite subtilité. Pour pouvoir mettre à jour les quatres éléments aux "coins", il faut disposer également d'une donnée de chaque processeurs voisin selon la diagonale. Mais cette donnée ne peut pas être transmise directement, à cause de la topologie de grille. Par contre, cette valeur est contenue dans une colonne échangée à un processeurs voisin de P lors de la première étape (voir figure 2), ce qui permet de demander directement la valeur aux processeurs voisins.

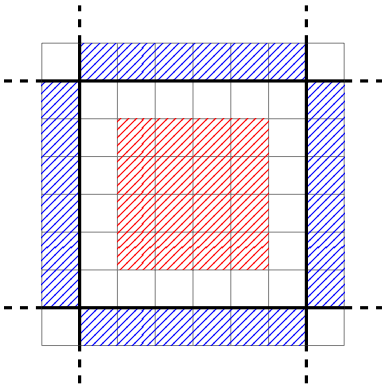

FIGURE 1 – Schéma des données d'un processeur. Les données en rouge peuvent être mises à jour par le processeurs, alors que les autres nécessitent les données en bleu des processeurs voisins.

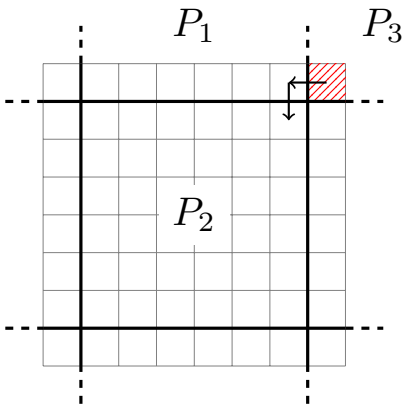

FIGURE 2 – Transmission des 4 données manquantes. La donnée (en rouge) qui intéresse P_2 est transmise à P_1 lors de la première phase où P_3 transmet sa colonne. P_2 peut donc demander cette données directement à P_1.

Question 3

L'algorithme final est présenté ci-dessous :

Data: Une matrice X
Result: La matrice $Y = \delta^+(X)$
p $\longleftarrow \sqrt{NbProcs()}$;
q $\longleftarrow$ MyNum();
N $\longleftarrow$ SizeOfGrid();
for *i = 1 to* $(\frac{N}{p} - 2)$ **do**
  for *j = 1 to* $(\frac{N}{p} - 2)$ **do**
    Mettre à jour l'élément $\widehat{x_{i,j}}$ avec les données de P_q.
  end
end
// send(les deux lignes et les deux colonnes) ;
// receive(les deux lignes et les deux colonnes) ;
for *j = 0 to* $(\frac{N}{p} - 1)$ **do**
  Mettre à jour les elements $\widehat{x_{0,j}}$ et $\widehat{x_{N-1,j}}$ avec les données reçues;
end
for *i = 1 to* $(\frac{N}{p} - 2)$ **do**
  Mettre à jour les elements $\widehat{x_{i,0}}$ et $\widehat{x_{i,N-1}}$ avec les données reçues;
end

Algorithm 1: L'algorithme de la question 3

Complexité

Soit ω le temps nécessaire pour effectuer une mise à jour d'un élément $x_{i,j}$. La première partie de l'algorithme (premières mises à jour, send et receive) prend un temps $max\left((\frac{N}{p}-2)^2\omega, 4(L+(\frac{N}{p}-2)b)\right)$.

La seconde partie prend un temps $(\frac{2N}{p}+2(\frac{N}{p}-2))\omega = (\frac{4N}{p}-4)\omega$.

D'où une complexité temporelle finale : $T(N,p) = max\left(\left(\frac{N}{p}-2\right)^2\omega, 4\left(L+\left(\frac{(N-2)b}{p}\right)\right)\right)+\left(\frac{4N}{p}-4\right)\omega$.

Remarque : Quand N tend vers $+\infty$, $T(N,p) \underset{N\to+\infty}{\sim} \left(\frac{N}{p}-2\right)^2\omega \underset{N\to+\infty}{\sim} \frac{N^2\omega}{p^2}$.

Adaptation sur une grille non torique

Si on conserve avec exactitude le principe de l'algorithme, le seul problème consistera pour les processeurs en bordure de grille, lors des envois et des réceptions. Concrètement, si on note cette fois $P_{i,j}$ les processeurs $((i,j) \in \{0,\dots,p-1\})$ selon leur place dans la grille, les processeurs $P_{0,i}$ ne peuvent pas envoyer leur ligne au processeur $P_{p-1,i}$ (même chose de $P_{i,0}$ vers $P_{i,p-1}$). Une solution pourrait être de les faire passer à travers toute une ligne (respectivement toute une colonne), ce qui nécessiterai un temps $(p-1)(L+\frac{(n-2)b}{p})$ supplémentaire pour chaque envoi, soit un temps total de $4p(p-1)(L+\frac{(n-2)b}{p})$. Toutefois ces envois peuvent se faire en parallèle d'autres calculs.

Et sur une topologie en anneau ?

Sur une topologie en anneau la situation se complique... Les transmissions verticales devront passer par exactement $p-1$ processeurs intermédiaires avant d'arriver à destination. Toutefois, on peut réutiliser la technique vue en cours pour le broadcast généralisé (où toutes les données transitent en même temps) pour les deux transmissions verticales que doivent accomplir chaque processeurs.

2 Average automata

Question 4

Voir le fichier `average.c`.

Question 5

Pour prouver que δ^+ est linéaire, il suffit de montrer que pour tous $(i,j) \in \mathbb{N}$, pour tout $k \in \mathbb{R}$,

$$(\delta^+(X+Y))_{i,j} = (\delta^+(X))_{i,j} + (\delta^+(Y))_{i,j} \quad \text{et} \quad (\delta^+(kX))_{i,j} = k(\delta^+(X))_{i,j}$$

.

Sachant que $(\delta^+(X))_{i,j} = (1-p)x_{i,j} + \frac{p}{4}(x_{i,j+1}+x_{i,j-1}+x_{i+1,j}+x_{i-1,j})$, le résultat est immédiat.

Principe de l'algorithme

L'idée est que si on a une fonction linéaire, on peut la représenter par une matrice, mais celle-ci sera de taille N^4.

Quelques éléments d'explications. Ici on identifie $\mathcal{M}_N(\mathbb{R})$ et $\mathbb{R}^{N^2}$ pour représenter la matrice X. La matrice X sous forme de colonne sera notée $\tilde{X}$.

$$X = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \qquad \longrightarrow \qquad \tilde{X} = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{pmatrix}$$

On peut alors voir la fonction δ^+ sous forme matricielle :

$$\Delta = \begin{pmatrix} p & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 & 0 \\ \frac{1-p}{4} & p & \frac{1-p}{4} & 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 \\ \frac{1-p}{4} & \frac{1-p}{4} & p & 0 & 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} \\ \frac{1-p}{4} & 0 & 0 & p & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & 0 & 0 \\ 0 & \frac{1-p}{4} & 0 & \frac{1-p}{4} & p & \frac{1-p}{4} & 0 & \frac{1-p}{4} & 0 \\ 0 & 0 & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & p & 0 & 0 & \frac{1-p}{4} \\ \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 & 0 & p & \frac{1-p}{4} & \frac{1-p}{4} \\ 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 & \frac{1-p}{4} & p & \frac{1-p}{4} \\ 0 & 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & p \end{pmatrix}$$

En effet :

$$\begin{pmatrix} p & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 & 0 \\ \frac{1-p}{4} & p & \frac{1-p}{4} & 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 \\ \frac{1-p}{4} & \frac{1-p}{4} & p & 0 & 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} \\ \frac{1-p}{4} & 0 & 0 & p & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & 0 & 0 \\ 0 & \frac{1-p}{4} & 0 & \frac{1-p}{4} & p & \frac{1-p}{4} & 0 & \frac{1-p}{4} & 0 \\ 0 & 0 & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & p & 0 & 0 & \frac{1-p}{4} \\ \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 & 0 & p & \frac{1-p}{4} & \frac{1-p}{4} \\ 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & 0 & \frac{1-p}{4} & p & \frac{1-p}{4} \\ 0 & 0 & \frac{1-p}{4} & 0 & 0 & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & p \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{pmatrix} = \begin{pmatrix} \delta(a) \\ \delta(b) \\ \delta(c) \\ \delta(d) \\ \delta(e) \\ \delta(f) \\ \delta(g) \\ \delta(h) \\ \delta(i) \end{pmatrix} = \delta^+ \left[\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{pmatrix} \right]$$

Nous avons donc exhibé une matrice Δ telle que $\Delta\widetilde{X^t} = \widetilde{X^{t+1}}$. Calculer X^t à partir de X_0 revient donc à calculer $\Delta^t\widetilde{X_0}$, ce qui peut se faire en temps $O(\log t)$ (avec N fixé) par exponentiation rapide. De plus, pour calculer les produits de matrice (ceux utiisés pour Δ^t et le produit $\Delta^+ \times X^0$), on peut utiliser l'algorithme vu en cours pour le produit matrice/matrice sur une grille de processeurs.

L'algorithme est donc le suivant :

Data: Une matrice X^0, un entier t
Result: La matrice X^t
Calculer la matrice Δ;
Calculer Δ^t via exponentiation rapide;
Calculer $\Delta^t \times X^0 = X^t$;
Retourner X^t.

Algorithm 2: L'algorithme de la question 5

Complexité en temps

Dans cette question, on pose $T_{prod}(N,p)$ le temps nécessaire pour calculer le produit de deux matrices de taille $N \times N$ sur une grille de $p \times p$ processeurs. On admet que l'on dispose d'un algorithme qui s'execute en temps $T_{prod}(N,p) = max\left(pL + N^2 b, \frac{N^3}{p}\omega\right)$ (un tel algorithme a été vu en cours).

Pour calculer la matrice Δ, il suffit juste d'initialiser une matrice avec N^4 éléments (car $\Delta \in \mathcal{M}_{N^2}(\mathbb{R})$). Cette initiatisation peut se faire de manière automatique car on a :

$$\Delta = \left(\begin{array}{c|c|c|c} A_N & B_N & \dots & B_N \\ \hline B_N & A_N & \dots & B_N \\ \hline \dots & \dots & \dots & B_N \\ \hline B_N & \dots & \dots & A_N \end{array}\right)$$

$$\text{Où :} \quad B_N = \frac{1-p}{4} \times I_N \quad \text{et} \quad A_N = \begin{pmatrix} p & \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} \\ \frac{1-p}{4} & p & \frac{1-p}{4} & \frac{1-p}{4} \\ \frac{1-p}{4} & \frac{1-p}{4} & p & \frac{1-p}{4} \\ \frac{1-p}{4} & \frac{1-p}{4} & \frac{1-p}{4} & p \end{pmatrix}$$

Pour l'exponentiation rapide, on sait qu'on effectue dans le pire des cas $\Theta(\log \text{t})$ produits de matrices, soit une complexité en $\Theta(T_{prod}(N^2,p) \times \log \text{t}) = \Theta\left(max\left(pL + N^4 b, \frac{N^6}{p}\omega\right) \times \log \text{t}\right)$ pour le calcul de Δ^t.

Enfin, le dernier produit matrice/vecteur se fait en temps $T_{prodVect} = max\left(pL + N^2 b, \frac{N^4}{p}\omega\right)$ (vu en cours, en utilisant $\sqrt{p}$ processeurs).

D'où une complexité temporelle finale de l'ordre de : $max\left(pL + N^4 b, \frac{N^6}{p}\omega\right) \times \log \text{t} + max\left(pL + N^2 b, \frac{N^4}{p}\omega\right)$.

Complexité en espace

Concernant la complexité en espace, chaque processeur dispose d'un bloc de Δ de taille $\left(\frac{N^2}{p}\right)^2$, d'un bloc de X^0 de taille $\frac{N^2}{p}$, mais également d'un autre bloc de taille $\left(\frac{N^2}{p}\right)^2$ nécessaire au calcul de Δ^t. Il ne faut pas non plus oublier que des blocs temporaires doivent être alloués pour les envois et réceptions de lignes et de colonnes durant les étapes où on effectue des multiplications.

Comparaison dans le cas général

Si on regarde les complexités temporelles,

Question 6

Si le réseau de processeurs forme une clique, l'algorithme précédent change uniquement dans la manière dont les communications sont effectuées.

Concernant les envois de données, ceux-ci peuvent être effectués sans passer par tout le réseau, chaque envoi aura donc un coût identique de $L + \frac{N^2}{p} b$.

Le calcul de Δ reste inchangé, sachant qu'il ne nécessite aucune communication entre processeurs.

Pour calculer Δ^t, on peut envoyer directement les blocs de matrices de la source à la destination. Sachant que pour calculer un bloc de taille $\left(\frac{N}{p}\right)^2$ il faut $2p$ blocs, le coût de cette opération est de

$2p \times \left(L + \left(\frac{N}{p} \right)^2 \right) b.$

Enfin, on utilise la même idée pour calculer le dernier produit matrice/vecteur,

Question 7

L'idée est d'utiliser la matrice Z^t afin d'éviter le calcul direct de X^t. On peut écrire X^0 sous la forme :

$$X^0 = \sum_{\substack{(i,j) \\ x^0_{i,j} \neq 0}} x^0_{i,j} E_{i,j}$$

où $E_{i,j}$ est la matrice dont tous les coefficients sont nuls, excepté le coefficient en (i,j) qui vaut 1. On remarquera que $Z = E_{0,0}$.

Comme δ^+ est linéaire, on a :

$$X^t = \sum_{\substack{(i,j) \\ x^0_{i,j} \neq 0}} x^0_{i,j} E^t_{i,j}$$

Cherchons maintenant comment calculer $E^t_{i,j}$. Vu que l'on travaille sur une grille torique, il est clair que $E^t_{i,j}$ et Z^t sont égales à une permutation près. Cette permutation constitue juste en un décalage des indices de la matrice.

Question 8

Voir le fichier `sparse.c`.

3 Réservoir thermique

Question 10

Question 11

Cette question est un peu ambigüe. L'additivité est clairement définie sur $\mathcal{M}_n(\mathbb{R})$, mais pas sur $\mathcal{M}_n(\mathbb{R} \cup \mathcal{C})$. En effet, il est difficile de définir le "type" (constant ou non) de la somme $x + \mathcal{C}(y)$. Si l'on considère que le résultat est une variable (cas 1), on perd le côté constant de y, alors que si on considère que le résultat est une constante (cas 2, le choix qui nous paraît le plus raisonnable), on perd le côté variable de x.

Mais quelque soit la convention que l'on choisisse d'adopter, l'application δ^+ n'est plus linéaire. En effet, considérons deux matrices X et Y telles que X soit nulle sauf pour $x_{0,0} = x \neq 0$, et Y soit nulle sauf pour $y_{0,0} = \mathcal{C}(y) \neq \mathcal{C}(0)$. On a clairement $\delta(x_{0,0}) = p \times x$ et $\delta(x_{0,0}) = y$.

Par contre, $\delta(x_{0,0} + y_{0,0}) = p \times (x + y)$ dans le cas 1, $x + y$ dans le cas 2. Dans tous les cas, ces deux valeurs différent de $p \times x + y$ si on suppose en plus que p est non nul. Ceci prouve que les matrices $\delta^+(X) + \delta^+(Y)$ et $\delta^+(X + Y)$ différent en leur élément $(0,0)$, ce qui montre bien que δ^+ n'est plus linéaire dans le cas où on rajoute des constantes.

Question 12

Question 13

Voir le fichier `constants.c`, très inspiré de `average.c`.

Question 14

Voir les fichiers `gfx.h` et `gfx.c`.