DM d'algo parallèle

Baptiste Rozière et Alice Pellet-Mary

26 novembre 2013

Question 3

Considérons p^2 processeurs organisés selon une topologie de carré torique de taille p*p et une matrice carrée de taille n*n.

Si p divise n, on répartit les cases de la matrices en assignant un bloc de taille $\frac{n}{p} * \frac{n}{p}$ à chaque processeur comme sur la figure ci dessous.

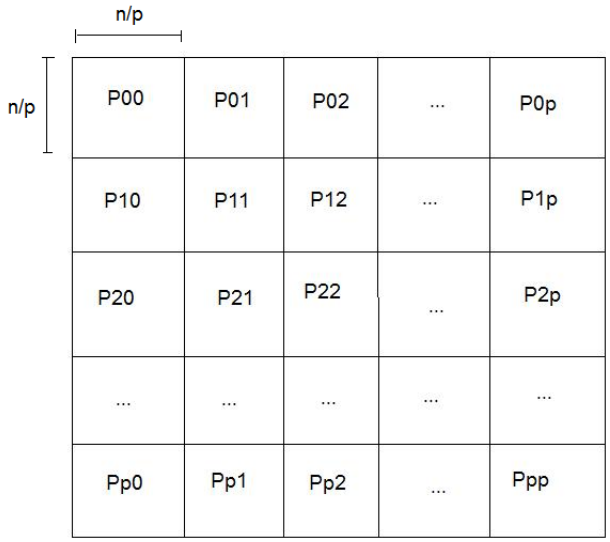

FIGURE 1 – répartition de la matrice sur les processeurs si p divise n

Dans le cas où p ne divise pas n, soit k = n mod p, on assigne tout simplement un rang de plus aux processus des k premiers rangs et une colonne de plus aux processus des k premières colonnes.

Chaque processus se charge de du calcul de la valeur suivante de toutes les cases de son bloc. Pour calculer les valeurs des cases situées aux extrémités, il a

besoin de connaître toutes les valeurs des cases de la matrice adjacentes à son bloc. Cela comprend $\frac{n}{p}$ valeurs pour chacun des processus situés au nord, à l'est, au sud et à l'ouest et une valeur pour les processus en diagonale donc $4(\frac{n}{p}+1)$ valeurs au total.

Chaque processus retiendra donc une matrice de taille $\frac{n}{p}+1 * \frac{n}{p}+1$, modifiera lui-même $\frac{n}{p} * \frac{n}{p}$ de ces cases et recevra celles situées aux extrémités des autres processus.

La topologie utilisée n'autorise pas à communiquer directement avec les processus diagonaux, ce qui représente 4 valeurs. Mais chacun des processus diagonaux envoie la valeur qui nous est utile à deux processus voisins du processus considéré sur la topologie. Par exemple, un processus a besoin de la valeur de la case 0,0 de la matrice du processus situé au sud-est et les processus sud et est ont tous les deux besoin de cette valeur pour calculer leurs cases $0, \frac{n}{p}$ et $\frac{n}{p}, 0$. Il est donc possible de l'envoyer en passant par l'un de ces deux processus sans surcoût de communication.

Question 5

L'automate est bien définit si on n'a pas dans DEP à la fois nord et sud, ou bien à la fois ouest et est. Si on a nord et sud ou est et ouest, on aura une boucle, chaque case devra se mettre à jour après ses voisins qui devront eux même se mettre à jour après elle. En revanche, si on n'a pas à la fois nord et sud ou à la fois est et ouest, l'automate sera bien défini car les lignes et colonnes des bords n'ont pas de voisins, donc elles pourront se mettre à jour sans attendre que d'autres cases se mettent à jour.
Si la première et dernière ligne sont voisines, on ne peut plus faire de dépendances nord ou sud. Les seuls cas où DEP sera bien défini seront $DEP = \{E\}$, $DEP = \{W\}$ ou $DEP = \emptyset$. Si on a N ou S dans DEP, par exemple N, la première ligne aura besoin de la dernière pour se mettre à jour, qui aura besoin de l'avant dernière, ..., qui aura besoin de la première. Donc l'automate n'est plus bien défini.
Dans le cas où la première et la dernière colonne sont voisines aussi, il n'y a plus aucune dépendance possible. Le seul cas où l'automate sera bien défini est le cas $DEP = \emptyset$.

Question 6

On peut trouver un algorithme efficace équivalent à l'algorithme séquentiel si les dépendances sont $DEP = \{N\}$, $DEP = \{W\}$ ou $DEP = \{N, W\}$. Dans les cas où on a S ou E dans les dépendances, on ne peut pas appliquer l'algorithme séquentiel, car comme on va de haut en bas et de gauche à droite, lorsqu'on veut mettre à jour une case, il n'y a que les cases au nord et à l'est qui ont été modifiées. On ne peut donc pas prendre en compte les modifications des cases à l'est et au sud.

Dans le cas où $DEP = \{N\}$, on a un algorithme parallèle efficace en pipelinant les calculs. On distribue la matrice entre les processeurs comme sur le dessin (un bloc de matrice par processeur).

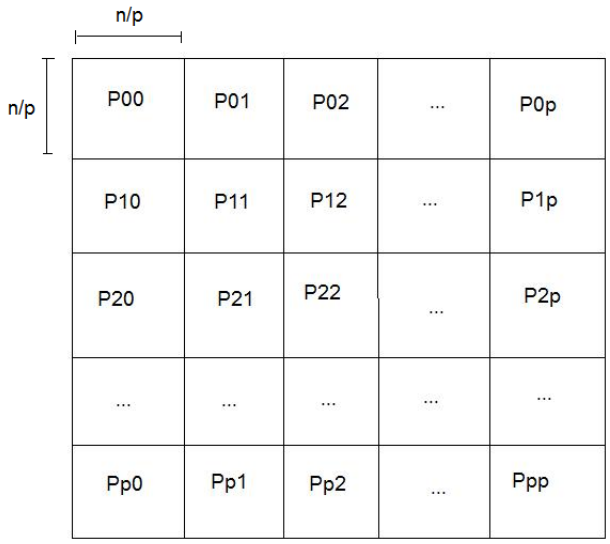

FIGURE 2 – répartition de la matrice sur les processeurs

Les blocs du haut mettent à jour la première ligne de leur bloc, et envoient le résultat de la dernière case de leur première ligne au bloc en dessous, qui peuvent alors commencer à calculer les valeurs de leur première ligne pendant que le bloc du dessus calcule les valeurs de sa deuxième ligne et ainsi de suite (cf figure 3).
Si on veut faire plusieurs mises à jours, les blocs du haut peuvent commencer la deuxième mise à jour une fois qu'ils ont fini de mettre à jour toutes leurs cellules. Comme ça les processeurs n'ont pas de moments oisifs à part au début lorsqu'ils attendent leurs premières données.
Si $DEP = \{E\}$, l'idée est la même.
Si $DEP = \{N, W\}$, on fait les calculs diagonales par diagonales (cf figure 4). On peut optimiser le temps de calcul en modifiant l'ordre de calcul dans les blocs, pour que les blocs voisins puissent commencer à travailler le plus tôt possible.

Si maintenant on a une topologie d'anneau, on peut toujours trouver des algorithmes efficaces. Par exemple si $DEP = \{N\}$, on attribue à chaque processeur un bloc de colonne, et ils font tous leurs calculs en parallèle sans avoir besoin de communiques (cf figure 5). De même si $DEP = \{W\}$ on attribue des blocs de lignes à chaque processeur.
Si $DEP = \{N, W\}$, on attribue des blocs de colonnes à chaque processeur, le

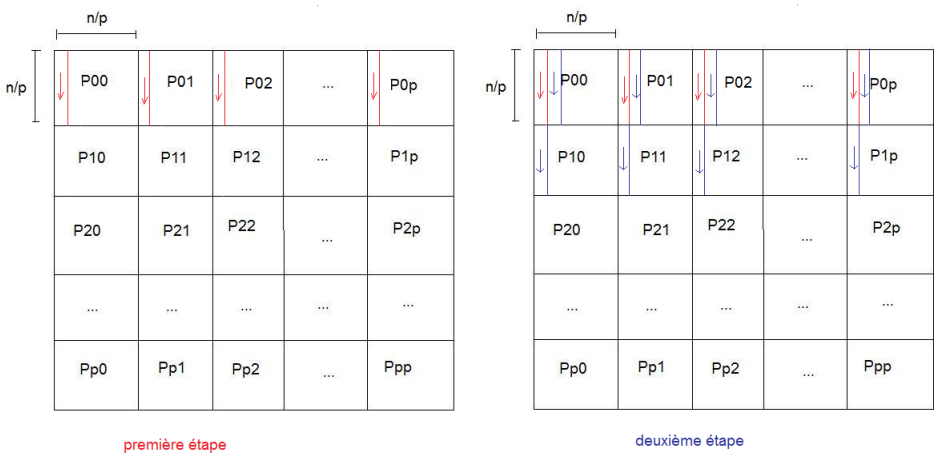

FIGURE 3 – mise à jour des éléments

premier calcule la première ligne puis envoie son dernier élément au deuxième, qui peut alors calculer sa première ligne, pendant que le première processeur calcule sa deuxième ligne et ainsi de suite (cf figure 6).

Question 8

On peut se ramener à un automate sans dépendances en découpant la matrice en lignes. On note $Ligne(i,t)$ la ième ligne de l'automate avec les dépendances après t épates de calcul (où une étape de calcul correspond à la mise à jour de tous les éléments de la matrice). Si la matrice initiale M est de taille nxn, on va considérer la matrice M' dont les 2 dernières lignes sont celles de M à l'instant 0, les 2 précédentes sont celles de M à l'instant 1, les 2 précédentes sont celles de M à l'instant 2, etc (cf figure).

Une fois qu'on a cette matrice, on peut mettre à jour toutes les lignes de rang impair simultanément, car la ligne au dessus d'eux est à l'instant d'après, et la ligne en dessous est au même instant. Donc tous les éléments autour sont dans le bon état pour calculer la mise à jour. On obtient la matrice de la figure 8.

Et on peut ensuite mettre à jour les lignes de rang pair simultanément. On réalise ainsi une mise à jour de l'automate en seulement 2 étapes. Pour se ramener à un automate où tous les éléments sont mis à jour à chaque étape, on considère des couples (i,j) où i est l'ancien élément de la matrice, et j indique si l'élément doit être mis à jour à cette étape ou pas (j alterne entre 0 et 1). Si on avait une fonction f permettant de mettre à jour une case en fonction de ses voisines dans l'automate initial ($f(a)$ représente la valeur de a après sa mise à jour en fonction

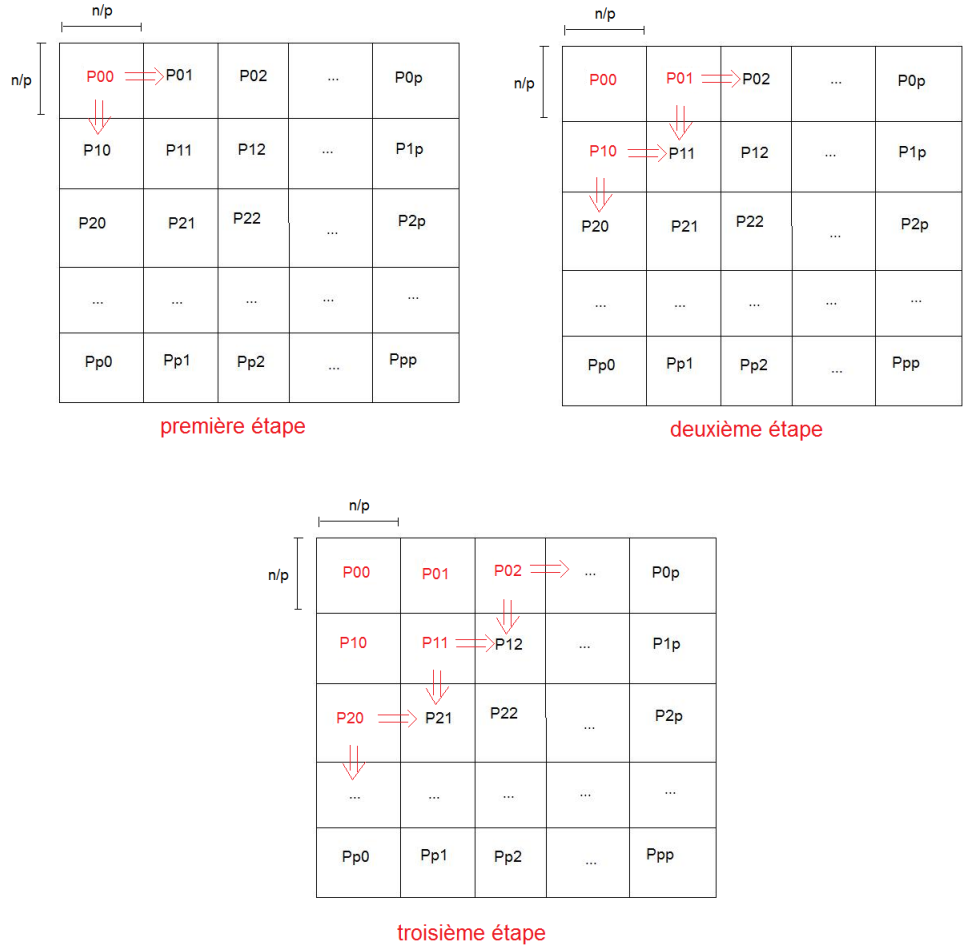

FIGURE 4 – $DEP = \{N, W\}$

de ses voisins), on définit une fonction g pour le nouvel automate : $g(a, 0) = (a, 1)$ (si $j = 0$, la case ne doit pas être mise à jour, mais elle devra l'être à la prochaine étape), et $g(a, 1) = f(a)$ (où $f(a)$ ne prend en compte que les premiers éléments des couples des cases voisines) (cf figure 9).

On obtient ainsi un automate sans dépendances équivalent à l'automate avec dépendances précédent.

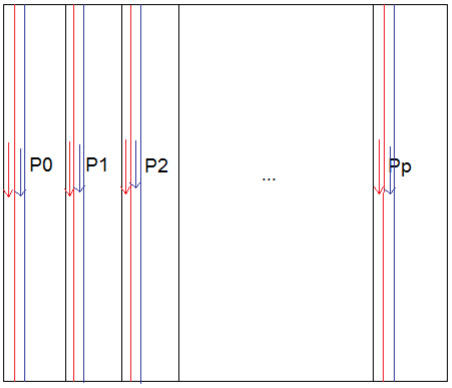

FIGURE 5 – topologie d'anneau et $DEP = \{N\}$

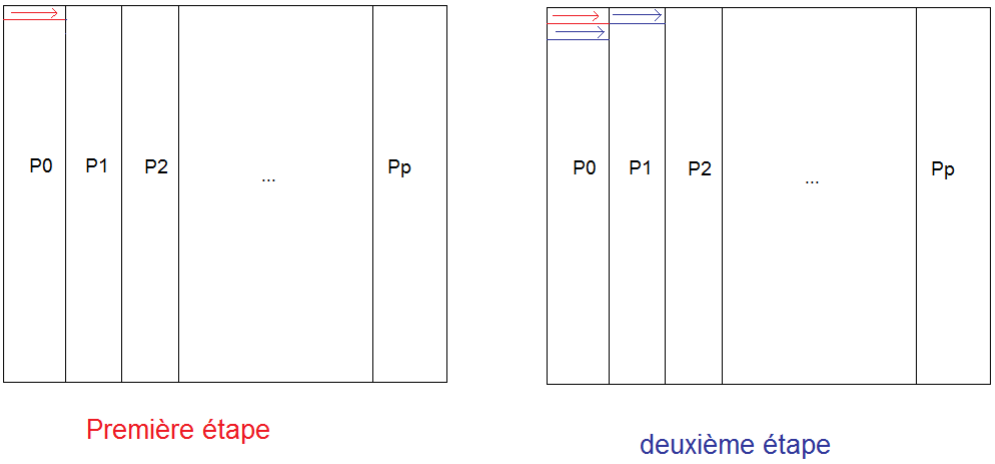

FIGURE 6 – topologie d'anneau et $DEP = \{N, W\}$

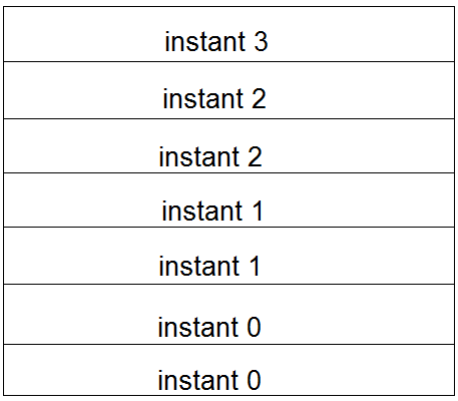

FIGURE 7 – assemblage des lignes dans le cas $n = 7$

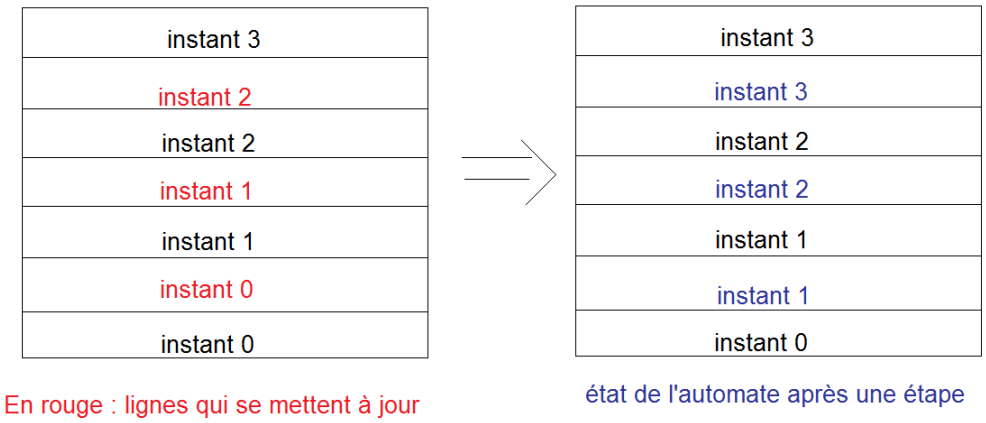

FIGURE 8 – une étape de calcul dans le cas $n = 7$

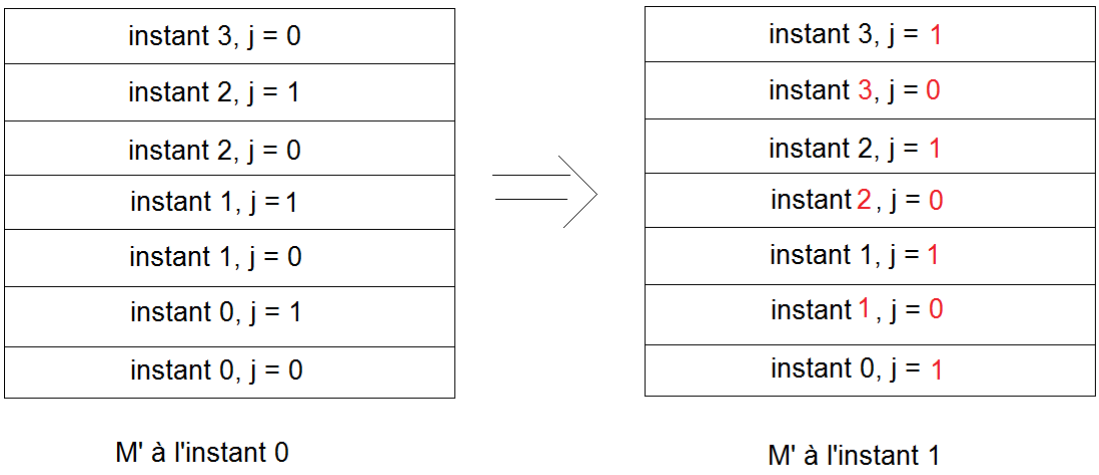

FIGURE 9 – une étape de calcul avec j