Master 1 d'informatique fondamentale
École Normale Supérieure de Lyon

Aloïs BRUNEL
Vincent DELAITRE

8 décembre 2008

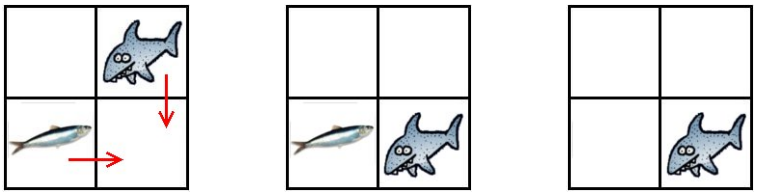
Devoir d'Algorithmique et Architectures Parallèles

Des sardines et des requins

1 Origamis de poissons

Question 1

Étant donné que tous les poissons ne peuvent pas être déplacés en même temps lors de la simulation, l'ordre de déplacement va être important. Par exemple, si un requin et une sardine souhaitent aller sur la même case, la sardine sera mangée si elle bouge en première et vivra dans le cas contraire.



(a) Configuration initiale

(b) Le requin bouge en premier

(c) La sardine bouge en première

Fig. 1: *Exemple où l'ordre de simulation importe*

Pour être plus formel, commençons par définir l'algorithme séquentiel de référence : on parcourt l'océan (qui correspond à un tableau à deux entrées, abscisse et ordonnée) ligne par ligne, de gauche à droite et de haut en bas. Par la suite, on va devoir s'assurer que l'algorithme parallèle se comporte comme en séquentiel. Cela signifie que si le déplacement extérieur des poissons est fixé par une entité extérieure, les algorithmes parallèle et séquentiel renverront le même résultat.

Question 2

Tout d'abord, commençons par décrire l'algorithme parallèle. Chaque processeur va simuler sa partie d'océan comme il le ferait en séquentiel : ligne par ligne, de gauche à droite et de haut en bas. La première colonne (celle où commencent le calcul d'une ligne en séquentiel) est éventuellement au milieu d'une zone. Pour raisonner sur l'algorithme, on fait comme si on coupait cette zone en deux au niveau de la première colonne. On ne perd pas en généralité en supposant que ces deux zones sont traitées séparément par deux processeurs et cela permet de définir le premier processeur : celui qui contient la première colonne, et le dernier processeur : celui qui est à gauche du premier processeur (cf figure 2).

Nous allons mettre en place un protocole de communication qui assure que l'ordre de simulation des cases de l'océan dans sa globalité respecte les conditions de la question précédente.

On cherche à minimiser le nombre de communications. Remarquons tout d'abord une chose : un processeur peut entièrement calculer une ligne sans effectuer de communication s'il a assez de place au début et à la fin de la ligne pour ne pas avoir à s'inquiéter des poissons provenant des processeurs voisins. Ainsi, un processeur ne peut faire aucune hypothèse sur l'avancement du calcul de ses voisins par rapport au sien. Il va donc falloir déterminer les différentes situations où il est nécessaire qu'un processeur demande à son voisin (qui est éventuellement en retard) si un poisson traverse la frontière ou non à telle ligne.

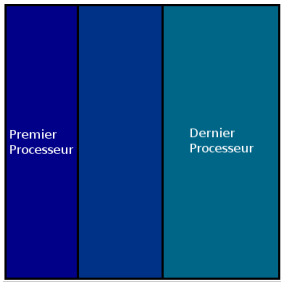


Fig. 2: *Définition du premier et du dernier processeur.*

On peut d'ores et déjà dégager la structure des communications du paragraphe ci-dessus. Les communications d'un processeur à un autre seront toutes constituées d'un numéro de ligne, d'un tag gauche/droite pour savoir si la requête vient du processeur de gauche ou de droite et de l'état de la case en bordure de la ligne (la case tout à gauche si on veut parler au processeur de gauche et inversement). L'état de la case indique simplement si la case contient un poisson et, le cas échéant, le type, l'âge, le degré de famine du poisson ainsi que sa volonté de traverser ou non la frontière.

Toutes les communications se font sous la forme de question/réponse pour une ligne donnée. Le processeur initialisant la communication envoie toutes les informations ci-dessus à son voisin et s'arrête de calculer en attendant la réponse. Si celui-ci a déjà traité la ligne objet de la requête, il peut répondre immédiatement en envoyant lui aussi les informations décrites dans le paragraphe précédent, sinon il doit effectuer le calcul de sa zone d'océan jusqu'à cette ligne.

Un processeur qui attend la réponse à sa question continue de répondre aux questions provenant de ses voisins (s'il peut y répondre). Par exemple si le processeur de gauche attend le processeur de droite qui est en retard, il doit être capable de répondre aux questions de celui-ci. Ainsi le processeur de droite va pouvoir rattraper celui de gauche et répondre à sa question.

Un processeur qui reçoit une requête pour une ligne qu'il n'a pas calculé ne peut pas y répondre tout de suite. Il la stocke, effectue le calcul jusqu'à cette ligne puis y répond. Cependant, le dernier processeur fait exception. Comme il est le seul processeur ayant un voisin de droite prioritaire (le premier processeur), il peut répondre immédiatement aux requêtes venant de la droite.

Voyons maintenant les différents cas qui nécessitent de se renseigner auprès de son voisin pour pouvoir commencer où finir le calcul d'une ligne.

- **Cas 1 :** Un poisson veut traverser la frontière entre deux processeurs. On effectue une communication avec le voisin concerné pour le lui indiquer. Lorsque celui-ci répond, chaque processeur connaît l'état complet des cases de part et d'autre de la frontière et peut donc décider si le poisson peut traverser et mettre à jour sa partie d'océan.

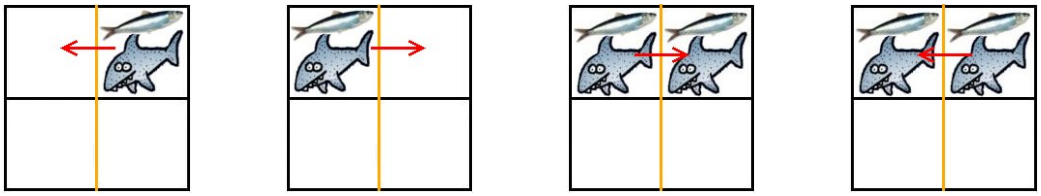


Fig. 3: *Toutes les possibilités du cas 1 : la barre orange est la frontière et la sardine et le requin dans la même case indique que le type du poisson n'importe pas*

– **Cas 2 (sauf pour le premier processeur) :** Le poisson est une sardine et se situe en bordure de la frontière gauche de la zone du processeur. On effectue une communication avec le voisin de gauche pour savoir si elle se fait manger par un requin. Après la réponse du voisin, chaque processeur connaît l'état des cases de part et d'autre de la frontière et agir en conséquence.

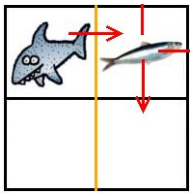
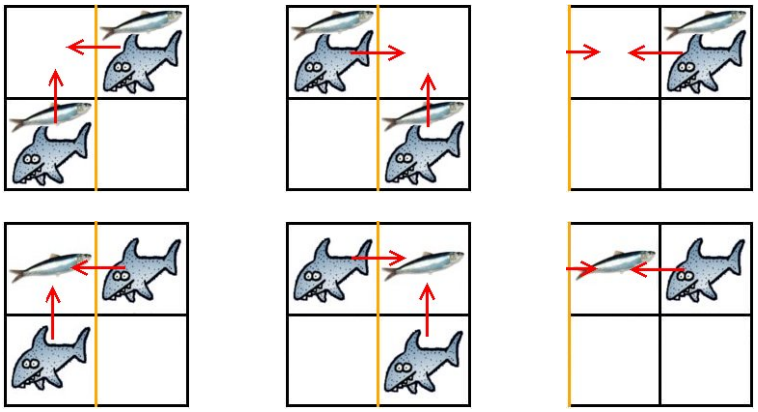


Fig. 4: *Cas 2 : la sardine peut éventuellement se faire dévorer (la barre orange est la frontière).*

– **Cas 3 :** Un poisson bouge vers une case située en frontière (c'est-à-dire le déplacement est autorisé, par exemple, le cas 3 s'applique pour un requin qui se déplacerait vers une case occupée par une sardine mais par pour un requin qui essaye de se déplacer vers un requin) et de priorité plus élevée que sa case initiale (ie située au-dessus ou à gauche). On effectue une communication pour s'assurer que la case de destination en frontière ne sera pas occupée entre-temps par un poisson venant d'un processeur voisin. Après la réponse, on décider si on a effectivement le droit de bouger le poisson.



(d) Requête de la gauche vers la droite

(e) Requête de la droite vers la gauche

(f) Requête de la droite vers la gauche

Fig. 5: *Toutes les possibilités du cas 3 : le premier processeur n'est pas concerné par les requêtes de la droite vers la gauche.*

Ces trois cas garantissent que l'on respecte l'ordre séquentiel, sous réserve que chaque processeur se souvienne de la configuration initiale de la colonne de gauche de sa portion d'océan et l'utilise pour pouvoir répondre correctement au processeur de gauche. Par exemple, imaginons qu'un requin sur la colonne de gauche s'est déplacé vers la droite. Si le processeur de gauche demande ensuite si une sardine peut passer à droite de la frontière (à l'endroit où était initialement le requin), il faut répondre que la case est occupée par le requin (et pas que la case est vide). Chaque processeur doit mettre à jour son souvenir de la colonne de gauche lorsqu'un poisson de cette colonne décide de descendre : le poisson doit être présent dans l'ancienne et la nouvelle case. Cela n'est pas valable pour le premier processeur (qui a priorité sur le processeur à sa gauche). Par contre, le dernier processeur doit se souvenir de sa colonne de droite (qui ne doit cependant pas être mise à jour lorsqu'un poisson descend) pour répondre correctement au premier processeur.

Ce protocole de communication ne peut pas bloquer. En effet, deux processeurs ne peuvent pas s'attendre mutuellement : le seul cas délicat est celui où les deux processeurs s'attendent sur la même ligne. Mais, dans ce cas le processeur de gauche est forcément dans le cas 1 ci-dessus. Il est en fin de ligne et peut répondre à la question du processeur de droite.

Ainsi, si on imagine qu'il y a blocage, alors les processeurs attendent soit tous leur voisin de gauche, soit tous leur voisin de droite. Mais le premier cas n'est pas possible car on a vu que le dernier processeur répond directement au premier processeur : ce dernier ne peut pas attendre son voisin de gauche. De même, le dernier cas n'est pas possible : il implique que tous les processeurs sont bloqués sur la même ligne de calcul et donc qu'ils attendent tous sur une requête de cas 1 à droite. Mais alors ils peuvent tous se répondre et ils ne sont pas bloqués.

Question 3

Pour obtenir un découpage optimal pour la charge de travail, on va utiliser la programmation dynamique. Soit n la taille de l'océan.

On se donne une fonction de poids f qui s'exécute sur une zone $[i,j]$ de l'océan pour $1 \leqslant i \leqslant j \leqslant n$, qui désigne la zone commençant à la colonne i et finissant à la colonne j. On définit $[i,j]$ pour $1 \leqslant j < i \leqslant n$ par $[i,j] = [i,n] \cup [1,j]$. On définit $\mathit{colonnes}([i,j])$ et $\mathit{poissons}([i,j])$ respectivement comme le nombre de colonnes et le nombre de poissons de $[i,j]$.

On définit l'ensemble des zones de l'océan :

$$\mathcal{Z} = \{[i,j] \ / \ 1 \leqslant i,j \leqslant n\}$$

ainsi que l'ensemble des p-découpages de $[i,j]$:

$$\mathcal{D}^p_{[i,j]} = \{(z_1, \cdots, z_p) \in \mathcal{Z}^p \ / \ [i,j] = \bigcup_{1 \leqslant k \leqslant p} z_k \text{ et } \forall k \neq k', z_k \cap z_{k'} = \varnothing\}$$

On étend f à l'ensemble des découpages : $\forall (z_1, \cdots, z_p) \in \mathcal{D}^p_{[i,j]}, f(x) = \sum_{1 \leqslant k \leqslant p} f(z_k)$.

On définit $\delta^f_{[i,j],p}$ comme étant le minimum des $\{f(x) \ / \ x \in \mathcal{D}^p_{[i,j]}\}$:

$$\delta^f_{[i,j],p} = \begin{cases} f([i,j]) & \text{si } p = 1 \\ \min\limits_{\substack{u \in [i,j],\, u \neq j \\ 1 \leqslant k < p}} (\delta^f_{[i,u],k} + \delta^f_{[u+1,j],p-k}) & \text{sinon} \end{cases}$$

On cherche à rendre la charge de chaque processeur la plus proche possible de la charge moyenne. Soit p le nombre de processeurs, et μ et ν des constantes représentant respectivement l'importance du nombre de colonne et du nombre de poissons dans le calcul de la charge (nous avons pris $\mu = n$ et $\nu = 100$: ainsi le poids d'une case est 1 et le poids d'un poisson est 100 ce qui est une bonne approximation des opérations nécessaires pour traiter une case et un poisson).

On définit la charge moyenne et la fonction f :

$$\text{ChargeMoyenne} = (\mu \cdot n + \nu \cdot \text{poissons}([1, n]))/p$$

$$f([i, j]) = \mu \cdot \text{colonnes}([i, j]) + \nu \cdot \text{poissons}([i, j]) - \text{ChargeMoyenne}$$

Ainsi, $\delta^f_{[1,n],p}$ est le minimum des écarts entre la charge des processeurs et la charge moyenne. Pour calculer une telle valeur, on utilise la programmation dynamique grâce à la définition récursive des $\delta^f_{[i,j],p}$ et on stocke les valeurs calculées dans un tableau à 3 dimensions (pour i, j et p) afin de ne pas les recalculer. L'algorithme fait $O(n^2 \cdot p)$ appels à f, qui s'exécute en $O(\text{colonnes}([i, j])) = O(n/p)$. La complexité de l'algorithme est donc en $O(n^3)$.

Pour obtenir le découpage optimal satisfaisant le minimum renvoyé par l'algorithme de programmation dynamique, il suffit d'utiliser le tableau de valeurs calculées pour retrouver de manière récursive les positions des frontières qui satisfont le minimum. On obtient de cette manière un découpage tel que celui-ci :

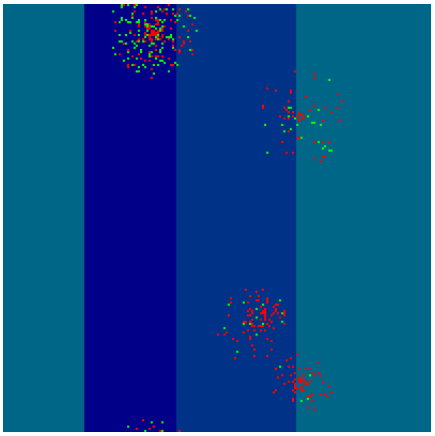


Fig. 6: *Zone 1 : 43 colonnes et 197 poissons, Zone 2 : 56 colonnes et 171 poissons, Zone 3 : 101 colonnes et 82 poissons*

Question 4

On peut réutiliser l'algorithme de la question précédente simplement en changeant la fonction d'évaluation f. Cette fois, on cherche à maximiser la distance moyenne entre les poissons et les bords de la zone (ie minimiser l'opposé de la distance moyenne). Pour $u \in [i, j]$, on note $D(u, [i, j])$ le minimum de la distance de la colonne u aux bords gauche et droit. On définit f :

$$f([i, j]) = \begin{cases} +\infty \text{ si poissons}([i, j]) = 0 \text{ et qu'il est possible d'attribuer un poisson par processeur} \\ -\min\limits_{u \in [i,j]} (D(u, [i, j])) \text{ sinon} \end{cases}$$

On peut ainsi connaître la configuration optimale qui maximise la distance moyenne des poissons aux bords de chaque zone. De plus, le cas spécial où la fonction f renvoie $+\infty$ garantit que chaque processeur aura au moins un poisson si c'est possible. Pour l'exemple précédent, on obtient la configuration suivante :

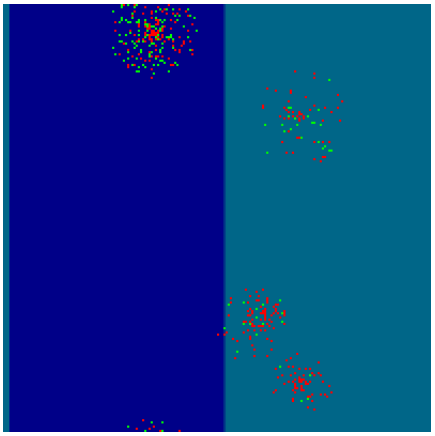


Fig. 7: *Zone 1 : 100 colonnes et 224 poissons (distance moyenne : 32), Zone 2 (au milieu de l'image) : 1 colonne et 2 poissons (distance moyenne : 0), Zone 3 : 99 colonnes et 224 poissons (distance moyenne : 27)*

Cependant, on ne sait pas dans quelle proportion maximiser la distance aux bords signifie minimiser le nombre de communications pour les k premiers tours (pour k fixé, nous avons choisi $k = n/p$). Typiquement dans l'exemple ci-dessus, la zone de largeur 1 au milieu de deux points à forte densité de poisson va engendrer beaucoup de communications.

Nous avons donc implanté une seconde méthode de découpage, qui renvoie un découpage correspondant à un minimum local pour le nombre de communications. L'idée est simple : si l'on sait évaluer le nombre moyen de communication engendré pour les k premiers tours pour un découpage donné, on va pouvoir faire évoluer un découpage initial petit à petit afin de minimiser le nombre moyen de communications. Il n'est pas possible d'utiliser la programmation dynamique car le calcul du nombre moyen de communication requière de connaître l'ensemble du découpage et on ne peut donc pas faire de fonction de poids s'appliquant uniquement à une seule zone comme précédemment.

Calcul du nombre moyen de communications : On se donne un découpage de l'océan, c'est-à-dire un tableau à p entrées (une par processeur) contenant chacune l'indice de la colonne de gauche et la largeur de la zone attribuée à un processeur. On cherche à calculer l'espérance du nombre de frontières traversées après k étapes de simulation pour un poisson situé initialement sur une colonne x donnée. Pour cela, on commence par précalculer un tableau représentant les frontières :

$$z(i) = \begin{cases} 1 & \text{si la colonne i commence ou termine une zone du découpage} \\ 0 & \text{sinon} \end{cases}$$

On note X la variable aléatoire désignant le nombre de frontières traversées, C celle désignant la colonne et Y celle désignant le nombre de d'étape de simulation. On a :

$$E(X/C = x/Y = k) = \sum_{i=0}^{\infty} i \cdot \mathbb{P}(X = i/C = x/Y = k)$$

$$\mathbb{P}(X = i/C = x/Y = 0) = 1, \text{ pour } i \leqslant 0$$
$$\mathbb{P}(X = i/C = x/Y = k) = 0, \text{ pour } i > k$$

Pour $i \leqslant k$:

$$\begin{aligned}\mathbb{P}(X = i/C = x/Y = k) = \quad & \tfrac{1}{4} \cdot (2 \cdot \mathbb{P}(X = i/C = x/Y = k-1) \\ & + \mathbb{P}(X = i - z(x+1)/C = x+1/Y = k-1) \\ & + \mathbb{P}(X = i - z(x-1)/C = x-1/Y = k-1))\end{aligned}$$

Alors :

$$\begin{aligned}E(X/C = x/Y = k) = \quad & \tfrac{1}{4} \cdot (2 \cdot E(X/C = x/Y = k-1) \\ & + \sum_{i=0}^{\infty} (i - z(x+1)) \cdot \mathbb{P}(X = i - z(x+1)/C = x+1/Y = k-1) \\ & + z(x+1) \cdot \sum_{i=0}^{\infty} \mathbb{P}(X = i - z(x+1)/C = x+1/Y = k-1) \\ & + \sum_{i=0}^{\infty} (i - z(x-1)) \cdot \mathbb{P}(X = i - z(x-1)/C = x+1/Y = k-1) \\ & + z(x-1) \cdot \sum_{i=0}^{\infty} \mathbb{P}(X = i - z(x-1)/C = x+1/Y = k-1))\end{aligned}$$

$$\begin{aligned}E(X/C = x/Y = k) = \quad & \tfrac{1}{4} \cdot (2 \cdot E(X/C = x/Y = k-1) \\ & + E(X/C = x+1/Y = k-1) + z(x+1) \\ & + E(X/C = x-1/Y = k-1) + z(x-1))\end{aligned}$$

Grâce à la formule ci-dessus, on peut calculer facilement par récurrence sur k l'espérance du nombre de frontières traversées pour un poisson situé sur une colonne x. L'espérance des communications après k coup est alors :

$$E(X/Y = k) = \sum_{x=1}^{n} E(X/C = x/Y = k) \cdot \text{poissons}([x, x])$$

Réajustement des zones : On sait désormais calculer le nombre moyen de communications pour un découpage donné. Définissons un bon découpage comme étant un découpage tel que chaque processeur ait au moins un poisson. Alors on veut trouver un bon découpage qui minimise ce nombre.

On génère tout d'abord un bon découpage à peu près homogène (chaque processeur reçoit une zone de taille environ n/p) commençant à la colonne d (la colonne de gauche de la première zone est la colonne d'indice d). Il suffit pour cela de vérifier que lorsque l'on attribue la i$^{\text{ième}}$ zone, elle contient au moins un poisson, sinon on augmente sa largeur. De plus, il faut qu'il reste assez de poisson pour les zones suivantes, sinon on diminue la largeur de la zone actuelle. Si un bon découpage existe, cet algorithme le trouve, sinon il revoit un découpage quelconque.

On fait varier l'indice d de la première colonne de 1 à n et on choisi le bon découpage qui minimise le nombre moyen de communications. On a ainsi un bon découpage qui correspond à peu près à la répartition des poissons. On va maintenant l'ajuster. Pour chaque zone du découpage, on essaie soit de la translater d'une colonne à gauche ou à droite, soit de lui rajouter une colonne à gauche ou à droite. A chaque fois, on vérifie que

ces transformations conservent la propriété de bon découpage et font diminuer le nombre moyen de communications (cela garantit par ailleurs que l'algorithme termine). On obtient ainsi un bon découpage qui est un minimum local pour le nombre de communications.

La complexité théorique de cet algorithme est assez délicate à mesurer car on ne sait pas exactement combien de réajustement (translations ou agrandissements) des zones on va effectuer. Supposons que l'on en fait $O(n)$ (largement suffisant, en pratique c'est moins). Alors on calcule l'espérance du nombre de communications en $O(n * k)$ (cf formule de récurrence). Pour un réajustement, on fait $O(p)$ appels au calcul d'espérance (à chaque fois qu'on modifie la zone d'un processeur). On obtient donc une complexité totale en $O(n^2 \cdot p \cdot k)$. Comme on a pris $k = n/p$, on retrouve la même complexité que pour la programmation dynamique : $O(n^3)$.

Pour la même distribution de poisson que celle utilisée par l'algorithme de programmation dynamique, on obtient le découpage suivant :

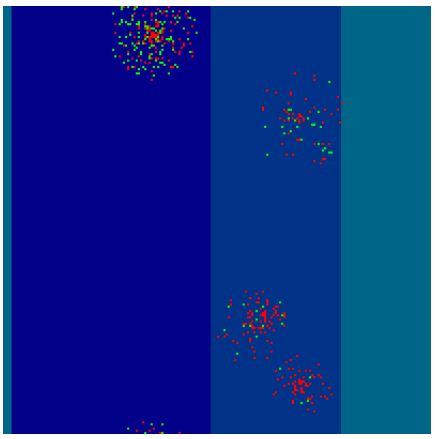


Fig. 8: *Zone 1 : 93 colonnes et 223 poissons (distance moyenne : 27), Zone 2 : 61 colonnes et 226 poissons (distance moyenne : 19), Zone 3 : 46 colonnes et 1 poisson (distance moyenne : 0)*

On voit que la somme des distances aux bords est plus faible que pour le cas de la programmation dynamique (46 contre 59) mais le résultat semble qualitativement meilleur. De plus il s'exécute expérimentalement plus rapidement (car on fait en fait peu de réajustement des zones).

Question 5

On cherche à optimiser le temps total de simulation de l'océan (calcul+communication). On pourrait utiliser la programmation dynamique en utilisant comme fonction de poids une combinaison bien choisie des fonctions de poids des questions 3 et 4. Cependant la fonction de poids de la question 4 calcule un espacement aux bords, et il va être difficile de mettre cette grandeur à l'échelle avec la charge processeur (c'est là toute la difficulté de la question).

On va donc plutôt utiliser le second algorithme de la question 4. On était capable de calculer le nombre moyen de communication. On n'a pas d'information sur la vitesse de communication mais on peut utiliser des valeurs standards de la latence et du débit pour avoir une estimation du temps de communication (par exemple celle d'une liaison Ethernet 10/100). De même on ne connaît pas le temps de calcul des processeurs mais on peut faire une estimation du temps de calcul maximum avec un processeur standard de 2GHz (le

temps de calcul maximum est atteint pour le processeur avec la plus grosse zone et le plus de poissons).

Ainsi, on peut lancer le second algorithme de la question précédente de manière à minimiser la somme des approximations du temps de calcul et du temps de communication. On aura ainsi un découpage équilibré pour une situation standard et il faudrait ajuster les trois constantes ci-dessus (latence et débit du réseau / vitesse du processeur) pour adapter le découpage à une situation précise.

Question 6

Lorsqu'un processeur a fini de simuler sa zone une fois, il s'arrête et attend les autres processeurs avant de simuler un nouveau pas de calcul. Lorsqu'il s'arrête, il signale à un processeur maître (disons le processeur 0) qu'il est prêt et envoie en même temps le temps de calcul (communication comprise) du dernier tour de simulation. Le processeur attend ensuite un signal du processeur maître pour relancer un nouveau pas de simulation. Ce signal est envoyé à tous les processeurs lorsqu'ils ont tous fini leurs calculs.

Pour détecter un déséquilibre dans le découpage, le processeur maître compare les valeurs maximum et minimum des temps de calcul : si la valeur maximum dépasse de $X_1\%$ la valeur minimum alors il décrète l'état de déséquilibre. La valeur $X_1\%$ doit être ajustée (on prend $X_1 = 10$) : trop élevée, on tolère des différences de temps de calcul importantes et trop faible, on redécoupe trop souvent. Dans les deux cas, cela dégrade les performances.

Lorsque l'état de déséquilibre est décrété, chaque processeur va envoyer son temps de calcul à ses voisins. Si le temps de calcul d'un processeur est plus élevé que le temps de son voisin, il lui donne une colonne de sa zone. Une fois que la passation des colonnes est terminée, chaque processeur signale à nouveau au processeur maître qu'il est prêt et ce dernier peut envoyer le signal de départ pour un nouveau pas de simulation.

L'état de déséquilibre est maintenu jusqu'à ce que la différence entre le temps maximum et le temps minimum passe en dessous de la barre des $X_2\%$ (on prend $X_2 = 5$). Cet algorithme converge vers un état d'équilibre, lentement cependant car les processeurs n'échangent qu'une colonne par tour. Nous voulions faire en sorte que le nombre de colonne passées au voisin dépende de la différence de temps de calcul entre les voisin. Mais comme il est difficile de relier une fraction d'océan au temps de calcul qu'elle implique (en terme de calcul et de communications), il était possible qu'un échange de plusieurs colonnes ne fasse qu'augmenter la différence de temps entre les deux processeurs (le processeur initialement en retard devenant en avance). Nous ne pouvions pas garantir que l'algorithme convergeait et nous avons préféré nous rabattre vers l'algorithme énoncé ci-dessus.

Question 7

L'algorithme global trouve un découpage qui minimise le temps de calcul, du moins c'est un minimum local, que l'on peut espérer proche du minimum global.

L'algorithme de redécoupage distribué impose que chaque processeur a une vue locale de la situation et permet d'assurer un équilibre local entre les voisins qui va petit à petit mener à un équilibre global. Cependant le prix du redécoupage n'est pas gratuit : il faut que chaque processeur transmette des colonnes à ses voisins si bien que l'on ne peut pas se permettre de faire un redécoupage à chaque simulation : on doit introduire une marge de tolérance de $X_1\%$, puis rééquilibrer jusqu'à ce que la différence des temps de calculs passe en dessous de la barre des $X_2 < X_1\%$.

L'exemple qui différencie le mieux l'algorithme global du local et celui où tous les poissons sont concentrés dans un coin de la carte. L'algorithme global va immédiatement équilibrer le nombre de poissons et le nombre de communication. L'algorithme local, quant à lui, va partir d'un découpage homogène si bien que de nombreux processeurs n'auront ni calculs ni communications à faire et un des processeurs devra faire tout le travail. Ce dernier va progressivement donner des colonnes à ses voisins, qui vont eux-mêmes en donner à leurs voisins et ainsi de suite jusqu'à évoluer lentement vers la situation d'équilibre trouvée par le découpage global.

Question 8

Le découpage par bande diminue le temps de calcul, du moins si le découpage est équilibré, ce que l'on peut supposer car le découpage par bande laisse plus de libertés que le découpage normal. Cependant le nombre de communications est doublé : on n'a plus seulement gauche/droite mais aussi haut/bas. Le découpage par bande donnera les mêmes performances qu'un découpage normal pour un réseau plus rapide mais des processeurs plus lents (et plus nombreux).

En termes d'effort de développement, cela induit plus d'effets de bords. De plus on peut désormais avoir plusieurs voisins de gauche et de droite. Le système de communication est donc bien plus compliqué (les voisins changent avec le redécoupage). Cela augmente aussi le nombre de cas à considérer (cf la question 2) pour savoir s'il est nécessaire ou non d'effectuer une communication. Cela implique de nombreux bugs et de nombreuses heures de prise de tête en plus.

Pour la détection des déséquilibres et le redécoupage, cela n'est pas forcément plus compliqué. On peut le faire en deux fois : d'abord on fait une détection du déséquilibre sur les colonnes et on redécoupe les colonnes comme précédemment, puis chaque colonne effectue une détection et un rééquilibrage sur ses lignes avec le même algorithme que pour les colonnes.

Le découpage par bande peut donc être implémenté, à condition d'y passer la nuit (et peut-être aussi celle d'après).