Algorithmique et techniques de base du calcul parallèle

Important. L'examen est composé de deux problèmes indépendants. Tous les documents sont autorisés.

Exercice A - Ordonnancement glouton

On considére une machine parallèle comportant p processeurs P_i $(0 \leq i < p)$. Chaque processeur connaît son indice $i \in 0, \ldots, p-1$.
De plus, le tableau de booléens `EstInactif[ 0 .. p-1 ]` donne l'état de chaque processeur: `EstInactif[ i ]` vaut 1 (respectivement 0) si le processeur i est inactif (respectivement actif).

Sur cette machine, on cherche à construire un algorithme parallèle de grain fin pour implanter un ordonnancement de type vol de travail.

1. On suppose que l'on a k tâches prêtes $J_1, \ldots, J_k$ qui peuvent être volées; ces tâches sont stockées dans une mémoire partagée accessible par tous les processeurs. On désire affecter à chaque processeur inactif j une tâche parmi $J_1, \ldots, J_k$ tel que deux processeurs inactifs différents aient deux tâches différentes. On calcule pour cela un tableau d'indices `JobCible[0 .. p-1]` tel que si le processeur j est inactif, alors il vole la tâche $J_{\text{JobCible}[j]}$.
   Donner un algorithme parallèle de coût $T_\infty = O(\log p)$ et $T_1 = O(p)$ qui calcule le tableau `JobCible`. **Indication:** on pourra utiliser un algorithme de calcul de préfixes.

2. On suppose maintenant que chaque processeur possède localement les tâches qu'il a créées. Le tableau en mémoire partagée `NbJobsLocaux[0 .. p-1 ]` donne le nombre de tâches prêtes sur chaque processeur. À un top t donné, pour $0 \leq i < p$, `NbJobsLocaux[i]` donne le nombre de tâches sur le processeur i:

   - si `NbJobsLocaux[i]`$=0$, le processeur i est inactif;
   - si `NbJobsLocaux[i]`≥ 1, le processeur i est actif; il possède alors `NbJobsLocaux[i]-1` tâches qui peuvent être volées par des processeurs inactifs.

   On désire calculer un tableau `IdentiteCible[ 0 .. p-1 ]` tel que si le processeur j est inactif, alors `IdenditeCible[ j ]` contient l'indice du processeur à qui il va voler une tâche. Deux processeurs inactifs peuvent voler un même processeur actif; mais un processeur actif ne doit pas être la cible de plus de processeurs qu'il n'a de jobs, i.e.

   $$\forall 0 \leq i < p \ : \qquad \text{Card}\,\{j = 0, \ldots, p-1 / \text{IdenditeCible}[j] = i\} \leq \text{NbJobsLocaux}[i] - 1$$

   Donner un algorithme parallèle de coût $T_\infty = O(\log p)$ et $T_1 = O(p)$ qui calcule le tableau `JobCible`. **Indication:** on pourra utiliser, outre des calculs de préfixes comme précédemment, des recherches dichotomiques.

3. Votre algorithme est-il EREW, CREW ou CRCW ? A quelle classe de complexité NC^k appartient ce problème d'allocation de tâches aux processeurs inactifs ?

Correction: *1. JobCible=PrefixePar(EstInactif).*

2. *Par j=0..p-1 if EstInactif[j] then NbAVoler[j] = 0 else NbAVoler[j] = NbJobsLocaux[j] - 1 ;*
   CumulNbAVoler = PrefixePar(NbAVoler) ;
   Par j=0..p-1 if EstInactif(j) {
   IdenditeCible[j]=RechercheDicho(JobCible[j], 0,p-1]) ; }
   Avec Recherchedicho(k,first,last) {
   if (first==last) return first;
   elseif (CumulNbAVoler[last] > k) return -1; //Pas assez de jobs
   else {
   m=(first+last)/2 ;
   if (CumulNbAVoler[m] < k) return Recherchedicho(k,first,m) ;
   else return Recherchedicho(k,m,last) ;
   }
   }
3. *L'algorithme est CREW à cause des accès concurrents au tableau CumulNbAVoler. Mais on peut le transformer en EREW (par coloriage déterministe comme dans le list ranking). Le problème est dans NC^1.*

Problème . Construction et analyse de programmes parallèles

On considère un programme dont le graphe de précédence est le graphe Diamant $n \times m$; on suppose $n \leq m$. Le schéma de calcul est décrit dans la figure 1.

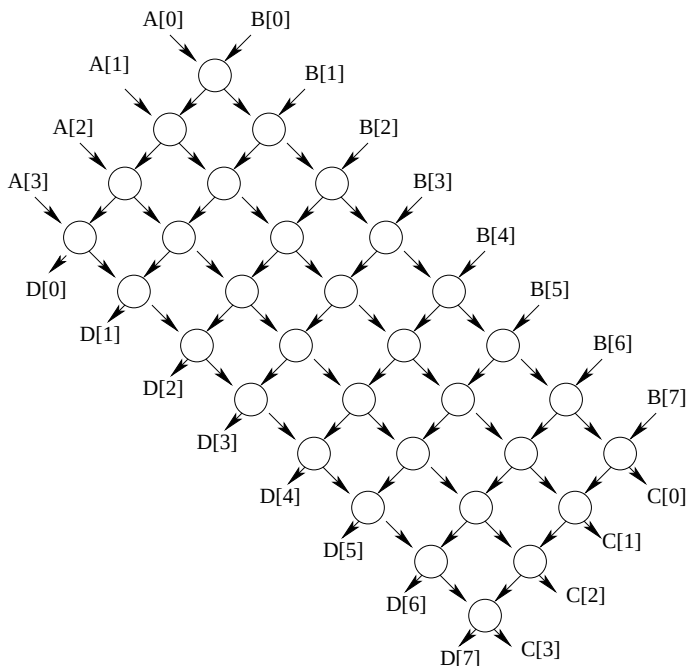

Figure 1: Graphe de précédence Diamant avec $n = 4$ et $m = 8$.

La procédure séquentielle de base est la suivante:

```
void DiamantSeq(int n, int m, IN Elt A[0 ..  n-1], IN Elt B[0 ..  m-1],
                              OUT Elt C[0 ..  n-1], OUT Elt D[0 ..  m-1] ) {

   Elt oldtmpA[0 ..  n-1] ; // zone de memoire pour calcul intermediaire;
   Elt newtmpC[0 ..  n-1] ; // zone de memoire pour calcul intermediaire;

   for (int k=0; k<n; k++) oldtmpA[k] = A[k] ;

   // Boucle de calcul
   for (int i=0; i<m; i++) {
      newtmpC[0] = localcompute ( oldtmp[0], B[i] ) ;
      for (int j=1; j<n; j++) {
         newtmpC[j] = localcompute ( oldtmpA[j], newtmpC[j-1] ) ;
      }
      D[i] = newtmpC[ n-1 ] ;
      for (int k=0; k<n; k++) oldtmpA[k] = newtmpC[k] ;
   }
   for (int k=0; k<n; k++) C[k] = newtmpC[k] ;
}
```

Dans toute la suite, on suppose que le côut de l'opération élémentaire `localcompute` est une constante τ ; on ne s'intéresse qu'au coût de ces opérations `localcompute`.

Analyse de l'algorithme séquentiel

On considère une machine SMP : on ne prend pas ici en compte le coût des communications.

Question I.

1. Calculer le coût T_1 en nombre d'opérations `localcompute` du graphe Diamant $n \times m$.
2. Calculer le chemin critique T_∞ du graphe Diamant.
3. Montrer que n processeurs suffisent pour atteindre ce temps T_∞. Dessiner sur le graphe un placement possible des calculs sur les processeurs.
4. On considère maintenant qu'on a p processeurs identiques avec n multiple de p. Comment regroupez-vous les calculs sur p processeurs ?
5. On considère maintenant qu'on a p processeurs différents mais ayant des vitesses proportionnelles; le processeur P_i a une vitesse v_i. Comment adaptez-vous le placement précédent ?

Correction:

1. $T_1 = n.m.\tau$
2. $T_\infty = (n + m - 1)\tau$
3. $\min(n, m) = n$ *processeurs suffisent. On fait un placement par diagonale (-appelée lignes dans la suite). Le processeur 0 fait les* m *tâches qui prennent les* $B[i]$ *en entrée; pour* $1 \leq i < n$, *le processeur* i *les tâches qui prennent en entrée les sorties du processeur* $i - 1$.
4. *On regroupe les processeurs virtuels cycliquement sur les p processeurs. Le processeur* i *exécute dans l'ordre les tâches allouées aux processeurs virtuels* i, $i + p$, ..., $i + p.(m/p)$.
5. *On normalise les vitesses en posant* $\nu_i = \frac{v_i}{\sum v_i}$. *Le processeur* i *simule alors* m/v_i *processeurs virtuels, toujours cycliquement.*

Prise en compte des communications

On s'intéresse maintenant au coût des communications. On suppose qu'un processeur ne peut émettre qu'un message à la fois; par contre, il peut calculer et recevoir un message pendant qu'il en émet un autre. On suppose que le délai de la communication d'une donnée de type `Elt` est α; i.e. lorsqu'un processeur P émet au top t une donnée de type `Elt` vers un processeur P', le processeur reçoit la donnée au top $t + \alpha$.

Question II.

1. Quels sont les volumes de communications C_1 et C_∞ dans le graphe Diamant ?
2. On suppose que l'on a n processeurs. Comment regrouperiez-vous les calculs ? Donner le coût d'exécution de ce placement en fonction de α, τ, n et m.

Correction:

1. $C_1 = n.m$ *et* $C_\infty = n + m$.
2. *Le processeur qui a la dernière dernière diagonale démarre au top* $(\alpha + \tau).(n - 1)$; *ensuite les communications sont recouvertes et on obtient le résultat en* $T_{exec} = (\alpha+\tau).(n-1)+m.\tau$

Algorithme parallèle par bloc

Dans toute la suite on suppose que $n = m$ et on s'intéresse maintenant à la prise en compte du coût de description du parallélisme (i.e. la description du graphe) sur machine SMP. On suppose que le coût de création d'un processus qui prend en entrée μ paramètres est μ.

Pour prendre en compte le surcoût induit par la génération du parallélisme (i.e. la description du graphe), on considère un algorithme `DiamantPar` basé sur un regroupement par blocs bidimensionnels des tâches. Pour cela, les tableaux A, B, C et D sont découpés en K blocs de (n/K) éléments; chaque calcul sur un bloc de taille n/K est effectué en séquentiel par l'algorithme DianmantSeq. La figure 2 illustre cette découpe pour $K = 2$.

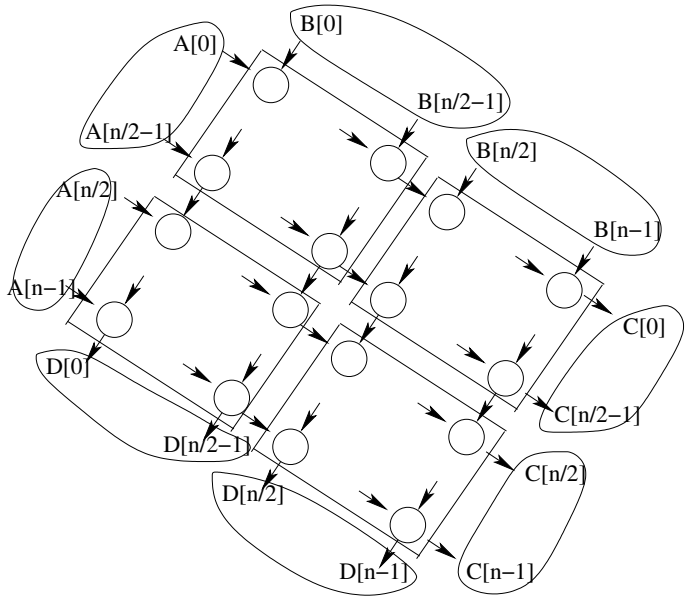

Figure 2: Diamant avec découpe des tableaux A,B,C,D en $K = 2$ blocs de taille $n/2$.

Question III. Algorithme par blocs sans coût de communication.
On ne prend pas ici en compte les coûts de communication.

1. Décrire (en pseudo-code) le schéma de l'algorithme par bloc; on précisera les paramètres de chaque processus.
2. Calculer le coût T'_1 de ce programme en fonction de K, n, τ et μ.
3. Calculer le temps T'_∞ de ce programme en fonction de K et n.
4. Quelle valeur choisissez-vous pour K ? Expliquez votre choix.

Correction: *1. for(i=0; i<n/K; i++) for (j=0; j<n/K; j++)*
fork DiamantSeq (n/K,m/K, A[i(n/K)..(i+1)n/K-1], B[...], C[...],D[...]) ;

2. $T'_1 = K^2.(4.\frac{n}{K})\mu + n^2\tau = O(1).\mu nK + n^2\tau$
3. *Le calcul du dernier bloc peut être effectué quand la description séquentielle de tous les blocs est terminée et lorsque tous les blocs qui le précèdent sur un chemin critique sont terminés. Il y a* $(2K-1)$ *blocs sur un chemin critique. D'où* $T'_\infty = n.K.\mu + (2K-1).\frac{n^2}{K^2}\tau \equiv= O(1)(n.K.\mu + \frac{n^2}{K}\tau)$
4. *Pour minimiser le temps d'exécution, on annulle la dérivée du temps par rapport à* K *et on obtient:* $(n\mu = \frac{n^2}{K^2}\tau)$*, d'où* $K = \frac{\tau}{\mu}\sqrt{n}$

Question IV. Algorithme par blocs avec coût de communication. On considère toujours que $n = m$ et on désire programmer l'algorithme par bloc de la question III sur une architecture distribuée avec des processeurs identiques. Afin de regrouper les communications, on ne communique un tableau résultat que lorsque le calcul de tous ses éléments est terminé. Le délai de la communication d'un bloc de n/K éléments est alors $\alpha + \beta.\frac{n}{K}$ et on suppose $\alpha \geq \beta$.

1. Donner le coût C_1 et C_∞ de l'algorithme par bloc en fonction de n, K, α et β.
2. En déduire le coût T_1 et T_∞ de l'algorithme par bloc en fonction de n, tau, α et β. Ce coût inclue les coûts de description du parallélisme et de communication
3. Comment choisissez-vous K pour minimiser le temps d'exécution ?
4. On suppose que le K optimal est K_{opt}, et on veut exécuter l'algorithme avec $p = K_{opt}$ processeurs. Comment placez-vous les tâches sur les processeurs ?

Correction: *1. Comme chaque procédure prend en paramètres $4n/K + 2$ données, on a $C_1 = K^2.(\alpha + \beta.4\frac{n}{K})$ et $C_\infty = 2K.(\alpha + 2.\beta.\frac{n}{K})$.*

2. On cumulle le temps de description du graphe, le temps de calcul de chaque bloc sur un chemin critique et le délai de communication entre 2 blocs. On obtient un temps d'exécution

$$T = n.K.\mu + \frac{n^2}{K}\tau + 2K(\alpha + 4.\beta.\frac{n}{K}).$$

3. Pour minimiser le temps, on annulle la dérivée du temps par rapport à K; on choisit donc $K_{opt} = \sqrt{\frac{n\mu+2\alpha}{\tau n^2}}$.

4. On effectue un ordonnancement au plus tôt; Il suffit d'avoir K_{opt} processeurs: top 0= tâche (0,0); top 1= tâches (0,1) et (1,0); top $K-1$, tâches (i,j) avec $i+j = K-1$. Puis on redescend; on a besoin de K processeurs.

Exercice B

On ce place dans les conditions du cours sur les communications collectives. Soit $p = 2^k$ processeurs. On considère que le coût de transmission d'un message de taille n entre deux processeurs est $t(n) = \alpha + n/\beta$. Un pocesseur ne peut émettre qu'un message à la fois. Par contre il peut émettre un message pendant qu'il reçoit un autre message.

1. Donner un algorithme de communication collective lorsque deux processeurs distincts doivent émettre chacun leur message vers tous les autres processeurs (2 broadcasts en parallèle).
2. Donner le coût de cet algorithme.
3. Reprendre les deux questions précédentes mais cette fois avec un algorithme adapté aux messages longs.
4. Justifier pourquoi ce second algorithme est plus performant que le précédent pour les messages longs (lorsque p est grand).

Correction: *On suppose que le processeur i veut envoyer son message M_i au processeur j qui a le message M_j. On suppose que $|M_i| \leq |M_j|$.*

1. *Le broadcast n'étant pas linéaire, 1 broadcast sur p processeurs coûte mons cher que 2 broadcasts sur p/2 processeurs. Aussi, on propose l'algorithme suivant:*
   - *étape 1. le processeur i envoie son message M_i au processeur j.*
   - *étape 2. le processeur j reçoit le message du processeur i et construit $M = M_i, M_j$.*
   - *étape 3. le processeur j broadcast M à tous les processeurs.*

2. *coût* $= \alpha + \beta|M_i| + broadcast(|M_i| + |M_j|)$...
3. *On fait un broadcast de messages longs (à compléter)...*
4. *Cela devrait apparaître clairement dans les formules (à compléter)...*