TP 2I003 Ordonnancement d'une application sur une machine hétérogène

Ce projet est a réaliser impérativement en binomes et à rendre à votre chargé de TD au plus tard le lundi 10 décembre 2018.

Le but de ce TP est d'étudier plusieurs algorithmes pour optimiser l'exécution d'une application donnée sous la forme d'une arborescence de tâches sur une machine hétérogène constituée de deux types de processeurs $\mathcal{A}$ et $\mathcal{B}$. Le nombre de processeurs de chaque classe n'est pas limité.

Notre application est modélisée par une arborescence $\mathcal{G} = (V, E)$ à n sommets numérotés de 0 à $n-1$. Tout sommet $v \in V$ est une tâche, tout arc $a = (u, v) \in E$ est une contrainte de précédence qui correspond à un transfert de données. Dans ce cas, v ne peut démarrer son exécution avant la fin de l'exécution de u.

D'une manière générale, pour tout couple de sommets $(u, v) \in V^2$, on a deux cas.

1. Si il existe un chemin dans $\mathcal{G}$ (cad une séquence d'arcs) de u vers v ou de v vers u, les deux tâches sont alors en relation de précédence. Si il existe un chemin de u vers v, alors v ne peut démarrer son exécution avant la fin de l'exécution de toutes les tâches du chemin de u vers v.
2. Dans le cas contraire, les deux tâches peuvent être exécutées en même temps, et cela indépendamment de l'affectation.

La racine est numérotée 0. Tout sommet différent de la racine 0 a exactement un prédécesseur, soit pour tout $u \neq 0$, il existe exactement un sommet v tel que $(v, u) \in E$. Pour tout sommet $u \in V$, $\mathcal{G}_u$ désigne la sous-arborescence de $\mathcal{G}$ constituée par u et l'ensemble de ses descendants dans $\mathcal{G}$. On note $\Gamma^+(u)$ l'ensemble des sommets $v \in V$ tels que l'arc $(u, v) \in E$.

Chaque tâche doit être exécutée sur un processeur de type $\mathcal{A}$ ou de type $\mathcal{B}$. Pour toute tâche $u \in V$, on note $t_{\mathcal{A}}(u)$ la durée d'exécution de u sur une machine de type $\mathcal{A}$. De même, $t_{\mathcal{B}}(u)$ désigne la durée d'exécution de u sur une machine de type $\mathcal{B}$. Une allocation des tâches est une fonction $\sigma : V \to \{\mathcal{A}, \mathcal{B}\}$ telle que $\sigma(u) = \mathcal{A}$ si la tâche u est allouée à une machine de type $\mathcal{A}$, et $\sigma(u) = \mathcal{B}$ si la tâche u est allouée à une machine de type $\mathcal{B}$.

Enfin, pour tout arc $a = (u, v) \in E$, on doit prendre en compte un temps de communication quand les tâches u et v ne sont pas exécutées sur des machines de même type. Soient alors $c_{\mathcal{AB}}(a)$ le temps de communication à considérer si $\sigma(u) = \mathcal{A}$ et $\sigma(v) = \mathcal{B}$. On note également $c_{\mathcal{BA}}(a)$ le temps de communication à considérer si $\sigma(u) = \mathcal{B}$ et $\sigma(v) = \mathcal{A}$. On pose alors $c_{\mathcal{AA}}(a) = c_{\mathcal{BB}}(a) = 0$. Le temps de communication à considérer pour une allocation σ pour a s'écrit alors $c_{\sigma(u)\sigma(v)}(a)$.

Ainsi, une instance $\mathcal{I}$ de notre problème est donnée par une arborescence $\mathcal{G} = (V, E)$, les durées $t_{\mathcal{A}}(v)$ et $t_{\mathcal{B}}(v)$ pour $v \in V$, et les temps de communications $c_{\mathcal{AB}}(a)$ et $c_{\mathcal{BA}}(a)$ pour $a \in E$.

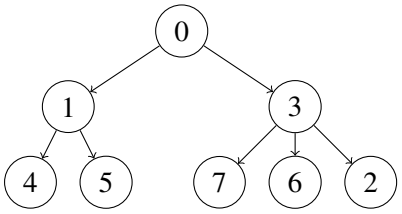

FIGURE 1 – Une arborescence $\mathcal{G}$ de racine 0

Un ordonnancement des tâches est défini par les dates $s(u)$, $u \in V$ d'exécution des tâches. La question posée dans ce TP est de calculer une allocation σ des tâches aux processeurs de sorte à obtenir un ordonnancement des tâches de durée minimum.

Supposons que σ soit fixé. Un ordonnancement s est réalisable pour σ si :

1. Pour tout $u \in V$, $s(u) \geq 0$.

$v \in V$	0	1	2	3	4	5	6	7
$t_{\mathcal{A}}(v)$	2	3	3	4	10	3	20	5
$t_{\mathcal{B}}(v)$	3	10	1	2	2	15	5	1

$e \in E$	$(0,1)$	$(0,3)$	$(1,4)$	$(1,5)$	$(3,7)$	$(3,6)$	$(3,2)$
$c_{\mathcal{AB}}(e)$	1	4	2	3	3	3	2
$c_{\mathcal{BA}}(e)$	4	3	4	2	4	3	3

TABLE 1 – Durées des tâches et des communications

2. Pour tout arc $a = (u, v) \in E$, on a alors deux cas. Si u et v sont exécutés par des processeurs de même type ($\sigma(u) = \sigma(v)$), alors v doit attendre la fin de l'exécution de u pour pouvoir s'exécuter. Sinon, il faut rajouter un temps de communication qui correpond au temps d'acheminement de la donnés d'une machine de type $\sigma(u)$ à une machine de type $\sigma(v)$. On obtient ainsi l'équation $s(u) + t_{\sigma(u)}(u) + c_{\sigma(u)\sigma(v)}((u, v)) \leq s(v)$.

Structures de données : Les structures de données sont stockées dans des tableaux. Plus précisément :

— L'arborescence est stocké dans un tableau `Arbre` de n entiers tel que, pour tout $u \in V$, `Arbre[u]` est l'unique sommet v tel que $(v, u) \in E$. Pour la racine, on pose `Arbre[0]=0`.

— Les durées des tâches sont stockées dans les tableaux de n entiers `tA` et `tB` se sorte que, pour tout $u \in V$, `tA[u]` et `tB[u]` sont respectivement la durée de u sur une machine de type $\mathcal{A}$ et $\mathcal{B}$.

— Les durées de communication sont stockées dans des tableaux de n entiers `cAB` et `cBA` tels que pour tout arc $a = (u, v) \in E$, la valeur $c_{\mathcal{AB}}(a)$ est stockée dans `cAB[v]`. De même, $c_{\mathcal{BA}}(a)$ est stockée dans `cBA[v]`.

— Une allocation σ peut être stockée dans un tableau de n éléments `Sigma` de 0 et de 1 telle que pour tout $u \in V$, `Sigma[u]` vaut 1 si $\sigma(u) = \mathcal{A}$.

On considère la définition inductive de l'ensemble $ABT(V)$ des arborescences sur l'ensemble V suivante :

Base : Les tâches isolées $u \in V$ forment des arborescences réduites à un sommet.

Induction : Soint $\mathcal{G}_{u_1}$, $\mathcal{G}_{u_2} \cdots \mathcal{G}_{u_p}$ une suite d'aborescences de $ABT(V)$ sans sommet commun, et un sommet $u \in V$ qui n'est dans aucune de ces arborescences. Alors $T = (u, \mathcal{G}_{u_1}, \mathcal{G}_{u_2}, \cdots, \mathcal{G}_{u_p}) \in ABT(V)$.

Exercice 1 – Implantation d'un premier algorithme

Question 1

On suppose dans cette question que l'allocation σ des tâches est fixée. Pour tout sommet $u \in V$, on pose $\Delta(u)$ la durée minimale d'un ordonnancement des tâches de $\mathcal{G}_u$ pour σ.

1. Calculez les valeurs $\Delta(u)$, $u \in V$ pour l'exemple et l'allocation $\sigma(0) = \sigma(1) = \sigma(5) = \mathcal{A}$ et $\sigma(2) = \sigma(3) = \sigma(4) = \sigma(6) = \sigma(7) = \mathcal{B}$.
2. En déduire sans démonstration des dates d'exécution $s(u), u \in V$ des tâches.
3. Montrez par induction structurelle sur $ABT(V)$ la propriété $\mathcal{P}(u)$, $u \in V$:

$$\Delta(u) = t_{\sigma(u)}(u) + \begin{cases} 0 & \text{si } \Gamma^+(v) = \emptyset \\ \max_{v \in \Gamma^+(u)}(\Delta(v) + c_{\sigma(u)\sigma(v)}((u, v)) & \text{sinon} \end{cases}$$

Indication : pour la partie inductive de la preuve, on suppose que, pour tout $v \in \Gamma^+(u)$, il existe un ordonnancement de $\mathcal{G}_v$ de durée minimale $\Delta(v)$.

Le but de cet exercice est d'implanter les structures de données utilisées et un premier algorithme qui calcule un allocation optimale et la durée de l'ordonnancement correspondant.

Question 2

1. Donnez les tableaux `Arbre`, `tA`, `tB`, `cAB` et `cBA` associés à l'exemple présenté en introduction.
2. Ecrire la fonction `duree(u, Sigmau, tA,tB)` qui retourne $t_{\mathcal{A}}(u)$ si `Sigmau` vaut 1, $t_{\mathcal{B}}(u)$ sinon. Quelle est sa complexité ? Justifiez votre réponse.
3. Ecrire la fonction `valCom(u,v, Sigmau,Sigmav,cAB,cBA)` qui retourne pour tout arc $(u, v) \in E$, 0 si les variables binaires `Sigmau` et `Sigmav` sont égales, $c_{\mathcal{AB}}((u, v))$ si `Sigmau` vaut 1, $c_{\mathcal{BA}}((u, v))$ sinon. Quelle est sa complexité ? Justifiez votre réponse.

Question 3
Ecrire la fonction `succ(u, Arbre)` qui pour toute tâche $u \in V$ retourne $\Gamma^+(u)$. Quelle est la complexité de cette fonction ? Justifiez votre réponse.

Question 4
Ecrire la fonction `CalculDelta(Arbre, tA, tB, cAB, cBA, Sigma, u)` qui, à partir de tous les paramètres d'une instance, une allocation `Sigma` et une tâche `u` calcule et renvoie $\Delta(u)$. Quelle est la complexité de cette fonction ? Justifiez votre réponse.

Question 5
Ecrire la fonction `CalculSigmaOptimum(Arbre, tA, tB, cAB, cBA)` qui calcule la durée d'un ordonnancement pour toutes les allocations possibles et choisit celle de durée minimale. Quelle est le nombre d'allocations possibles ? En déduire la complexité de cette fonction.

Indication : Pour tout $i \in \{0, \cdots, n\}$, posons $x_i^\sigma = 1$ si $\sigma(i) = \mathcal{A}$, $x_i^\sigma = 0$ sinon. Soit également S l'ensemble des allocations possibles pour n tâches. Alors, on remarque que la fonction $f : S \to \{0, \cdots 2^n - 1\}$ définie par $f(\sigma) = \sum_{i=0}^{n-1} x_i^\sigma 2^i$ est une bijection. Il suffit donc de calculer pour tout $\alpha \in \{0, \cdots 2^n - 1\}$, la décomposition de α en base 2 pour obteniir toues les allocations possibles.

Exercice 2 – Implantation d'un second algorithme

Le but de cet exercice est d'implanter un second algorithme pour calculer une allocation σ de durée minimale.
Pour toute tâche $u \in V$, on note $D^{\mathcal{A}}(u)$ et $D^{\mathcal{B}}(u)$ les deux suites définies par :

$$D^{\mathcal{A}}(u) = t_{\mathcal{A}}(u) + \max_{v \in \Gamma^+(u)} \min(D^{\mathcal{A}}(v), c_{\mathcal{AB}}((u, v)) + D^{\mathcal{B}}(v))$$

et

$$D^{\mathcal{B}}(u) = t_{\mathcal{B}}(u) + \max_{v \in \Gamma^+(u)} \min(D^{\mathcal{B}}(v), c_{\mathcal{BA}}((u, v)) + D^{\mathcal{A}}(v)).$$

Pour tout allocation σ, on définit $\bar{\sigma}$ par, pour tout $u \in V$:

$$\bar{\sigma}(u) = \begin{cases} \mathcal{A} & \text{si } \sigma(u) = \mathcal{B} \\ \mathcal{B} & \text{sinon.} \end{cases}$$

Soit alors l'allocation σ définie par :

$$\sigma(0) = \begin{cases} \mathcal{A} & \text{si } D^{\mathcal{A}}(0) \leq D^{\mathcal{B}}(0) \\ \mathcal{B} & \text{sinon.} \end{cases}$$

et pour toute tâche $v \neq 0$ avec $(u, v) \in E$,

$$\sigma(v) = \begin{cases} \sigma(u) & \text{si } D^{\sigma(u)}(v) < D^{\bar{\sigma}(u)}(v) + c_{\sigma(u)\bar{\sigma}(u)}(u, v) \\ \bar{\sigma}(u) & \text{sinon.} \end{cases}$$

Question 1
Calculez les valeurs $D^{\mathcal{A}}(u)$ et $D^{\mathcal{B}}(u)$, $u \in V$ pour l'exemple donné. En déduire une allocation σ.

Question 2
Démontrez par induction structurelle sur $ABT(V)$ que, pour tout sommet $u \in V$, $D^{\mathcal{A}}(u)$ et $D^{\mathcal{B}}(u)$ est respectivement une borne inférieure de la durée d'un ordonnancement réalisable de $\mathcal{G}_u$ avec $\sigma(u) = \mathcal{A}$ et $\sigma(u) = \mathcal{B}$.

Pour cela, on note pour tout $u \in V$, $\Delta^{\mathcal{A}}(u)$ la durée minimale d'un ordonnancement optimal qui impose $\sigma(u) = \mathcal{A}$. De même, on note $\Delta^{\mathcal{B}}(u)$ la durée miniumale d'un ordonnancement optimal qui impose $\sigma(u) = \mathcal{B}$. On démontre alors que $\Delta^{\mathcal{A}}(u) \geq t_{\mathcal{A}}(u) + \min(\Delta^{\mathcal{A}}(u_q), \Delta^{\mathcal{B}}(u_q) + c_{\mathcal{AB}}((u, u_q))$, puis on utilise cette inégalité pour démontrer la propriété.

Question 3
Ecrire une fonction `CalculBorneinf(Arbre, tA, tB, cAB, cBA)` qui calcule en $\Theta(n)$ l'ensemble des valeurs $D^{\mathcal{A}}(u)$ et $D^{\mathcal{B}}(u)$, $u \in V$ qui seront alors stockées dans des tableaux. Justifiez votre réponse pour la complexité.

Remarque : les tableaux $D^{\mathcal{A}}$ et $D^{\mathcal{B}}$ peuvent être passés en paramètre au besoin et modifiés par effet de bord.

Question 4
Ecrire la fonction `CalculSigmaOptimum2(Arbre, tA, tB, cAB, cBA, DeltaA,DeltaB)` qui à partir des valeurs $D^{\mathcal{A}}(u)$ et $D^{\mathcal{B}}(u)$, $u \in V$ stockées dans un tableau calcule une allocation de durée minimale σ en $\Theta(n)$. Justifiez votre réponse pour la complexité.

Remarque : le tableau σ peut être passé en paramètre au besoin et modifiés par effet de bord.

Exercice 3 – Mesure expérimentale de la complexité

Le but de ce dernier exercice est de mesurer de manière expérimentale la complexité du calcul d'une allocation de durée minimale et la durée correspondante. Le choix du langage de programmation est libre. La complexité expérimentale de l'exécution d'une fonction correspond ici au temps qu'il faut pour que la fonction s'exécute.

Question 1
Programmez et testez l'ensemble des fonctions demandées. Fournir un jeu d'essai pour les fonctions `CalculSigmaOptimum`, `CalculBorneinf` et `CalculSigmaOptimum2`.

Question 2
Pour simplifier la structure des arbres, on va effectuer l'ensemble des expérimentations sur des arbres binaires complets. Pour $h > 0$ fixée, il s'agit de l'unique arbre parfait h tel que tous les niveaux sont pleins, y compris le dernier. Le nombre de tâches est alors $n = 2^h - 1$. De plus, en numérotant les sommets de manière similaire au cours, mais en démarrant à 0, on observe que le prédédecesseur du sommet $i \in \{1, \cdots, n-1\}$ est $\lfloor \frac{i+1}{2} \rfloor - 1$.

Ecrire la fonction `CalculArbreComplet(hauteur)` qui renvoie un arbre de complet de hauteur h. Ecrire la fonction `InstanceAleatoire(n,M)` qui caculle toutes les valeurs numériques liées à une instance de n tâches en tirant des entiers compris entre 1 et M.

Pour les expérimentations, on pourra se fixer une valeur de M, par exemple $M = 50$.

Question 3
On souhaite dans cette question étudier les limites d'une part de la fonction `CalculSigmaOptimum`, d'autre part des fonctions `CalculBorneinf(Arbre, tA, tB, cAB, cBA)` et `CalculSigmaOptimum2`. Pour cela, exécutez les pour des arbres de hauteur h de plus en plus importante. Quelle est la la plus grande valeur de h que vous pouvez traiter (sans problème de mémoire, et/ou en un temps raisonnable de quelques minutes) ? Donnez la valeur de n correspondante.

Question 4
On souhaite dans cette question vérifier de manière expérimentale que la fonction `CalculSigmaOptimum` a une complexité exponentielle.

1. Testez la fonction pour des tailles de séquence $n = 2^h - 1$ de plus en plus grandes. Soit $T_1(n)$ le temps que vous obtenez pour une instance de $n = 2^h - 1$ tâches. Donnez les valeurs $T_1(n)$ obtenues.

2. Vérifiez expérimentalement que $\log T_1(n)$ est une fonction linéaire de n (pour les valeurs importantes de n) en calculant la pente de la droite.

Question 5

On souhaite maintenant vérifier de manière expérimentale que les fonctions `CalculBorneinf(Arbre, tA, tB, cAB, cBA)` et `CalculSigmaOptimum2`. ont une complexité linéaire.

1. Testez la fonction pour des valeurs $n = 2^h - 1$ de plus en plus grandes. Soit $T_2(n)$ le temps global d'exécutions de ces deux fonctions pour une instance de $n = 2^h - 1$ tâches. Donnez les valeurs $T_2(n)$ obtenues.
2. Vérifiez expérimentalement que $\frac{T_2(n)}{n}$ est une fonction constante quand $n = 2^h - 1$ croît (pour les valeurs importantes de n) en calculant cette valeur.