Algorithmique combinatoire sur ordinateurs paralleles

Ivan Lavallee

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(1) 39 63 55 11

Rapports de Recherche

N° 637

ALGORITHMIQUE COMBINATOIRE SUR ORDINATEURS PARALLÈLES

Ivan LAVALLÉE

Mars 1987

ALGORITHMIQUE COMBINATOIRE SUR ORDINATEURS PARALLELES

PARALLEL COMPUTERS AND COMBINATORIAL ALGORITHM

Ivan LAVALLEE

Résumé

Dans ce travail, nous étudions l'impact du parallélisme sur la conception d'algorithme dans le domaine combinatoire, et ce en traitant deux exemples. Le premier exemple est l'élaboration d'un algorithme parallèle pour extraire un arbre couvrant minimal dans un très grand graphe. Ce problème est bien connu pour être aisément calculable (i.e. dans $\mathcal{P}$), et notre propos était de voir ce que le parallélisme peut apporter dans un tel cas.

Le deuxième exemple est la parallélisation des méthodes de recherche arborescente, méthodes qui sont essentiellement utilisées lorsqu'on veut résoudre des problèmes difficilement calculables (i.e. $\mathcal{NP}$ complets).

Nous discutons l'étude et montrons quelques liens existants entre algorithmes parallèles et séquentiels en montrant comment on peut déduire un nouvel algorithme séquentiel à partir d'une remarque sur des "anomalies" dans l'énumération implicite parallèle. On termine en essayant de faire émerger des idées permettant de concevoir des algorithmes spécifiques du parallélisme.

Abstract

In this work we study why and wherefore we must use parallel computers in computation of combinatorial algorithms. We have chosen to show how this purpose can be achieved with two examples. The first one is the design of a parallel algorithm which builts a Minimum Spanning Tree on a very large graph. This problem is known as a tractable one and our purpose is to see how it is possible to obtain better performances in parallel environment than in sequential environment.

The second example that we discuss is a general way in implicit ennumeration area, it is the Branch and Bound. The problems which are solved by this method are untractable problems.

We show how these problems are stated in parallel terms and what are the new ideas which consecutively arise.

ALGORITHMIQUE COMBINATOIRE SUR ORDINATEURS PARALLELES

Les super-ordinateurs ont, pour l'essentiel, été conçus et utilisés jusqu'ici pour résoudre des problèmes numériques. A tel point qu'on a pu appeler ces machines des "souffleries numériques". La capacité de traitement de ces machines est telle qu'on a gagné plusieurs ordres de magnitude dans le nombre d'opérations traitées par seconde. On peut maintenant aborder des problèmes de simulation auxquelles on n'aurait osé songer il y a seulement dix ans.

Cependant, la taille des problèmes numériques traités est sans commune mesure avec les tailles que peuvent atteindre certains problèmes combinatoires.

Nous avons donc testé l'utilisation de telles machines pour résoudre des problèmes combinatoires. Le but étant non seulement d'apprécier le gain de performance obtenu lors du passage de séquentiel en parallèle d'un programme donné, mais surtout d'essayer de réfléchir sur la question de l'algorithmique parallèle proprement dite.

En effet, les grandes questions qui se posent tournent autour du problème de la définition du terme algorithme en environnement parallèle, ainsi que des performances des dits algorithmes.

Ainsi, il existe une idée largement répandue qui s'exprime comme suit :

si un calcul est réalisable en un temps t avec un processeur, et en un temps t' avec q processeurs, alors on a toujours :

$$\frac{t}{t'} \leq q$$

Cette idée est fausse comme nous le verrons par la suite. De même, l'augmentation du nombre de processeurs peut, dans certains cas conduire à un effondrement des performances, c'est à dire ici, à avoir t" > t', t" étant le temps obtenu pour le même calcul avec q+1 processeurs.

De même, il convient, lors du calcul des performances attendues d'un algorithme, de bien prendre en compte les particularités induites par l'architecture de machine considérée (voir [LAV1]).

Dans ce qui suit, nous avons étudié l'algorithmique de deux problèmes dont nous avons ensuite programmé la résolution sur un CRAY-XMP4 (pour d'autres problèmes de combinatoire traités en parallèle, on peut aussi consulter [COR] et [FRA]).

Les deux problèmes étudiés sont :

- la recherche d'un arbre couvrant de poids minimal dans un très grand graphe (M.S.T.)
- l'énumération implicite (B & B).

Le premier de ces deux problèmes, bien connu, ressortit à la théorie des graphes et à la recherche opérationnelle, c'est un problème de nature polynomiale - déterministe.

Pour l'énumération implicite, il s'agit plutôt d'une méthode générale de résolution pour une classe de problèmes non polynomiaux - déterministes comme par exemple, le problème du voyageur de commerce, le problème des tournées, la programmation linéaire en variables binaires, l'affectation quadratique etc...

1. L'ARBRE COUVRANT MINIMAL

Le problème de la détection d'un arbre couvrant de poids minimum dans un graphe valué est un problème très important, à la fois en recherche opérationnelle et en informatique.

Du point de vue de la recherche opérationnelle l'intérêt du problème, outre bien entendu le fait qu'il est un problème réel (c'est là l'objet premier de la Recherche Opérationnelle), réside en la façon de le résoudre et les algorithmes ainsi engendrés. Ce type d'algorithme est dit "glouton" et le substrat théorique est lié à la théorie des matroïdes [BER] [FOU].

Avec une approche séquentielle il existe deux grandes familles d'algorithmes pour résoudre ce problème, l'une est basée sur un algorithme dû à Sollin (voir [SOL]), l'autre sur la théorie des matroïdes et les travaux de Kruskal, [KRU]. L'algorithme de Sollin repose implicitement sur une propriété locale que nous mettons en évidence car elle permet de construire des algorithmes parallèles et des algorithmes distribués.

La famille des algorithmes basés sur cette propriété locale comprend outre l'algorithme de Sollin, essentiellement les algorithmes de Prim (voir [PRI]).

L'autre famille est fondée sur les travaux de Kruskal [KRU] et s'appuie d'abord sur un traitement global des arêtes du graphe initial ; cette famille d'algorithmes repose sur les concepts de la théorie des matroïdes [BER], [FOU].

Du point de vue informatique, l'intérêt du problème provient du fait que la structure d'arbre est une topologie efficace pour les problèmes qui se posent dans les réseaux d'ordinateurs. En effet, la structure "en arbre" infère qu'il existe une chaîne et une seule d'un point à un autre du réseau. Si on particularise l'un des noeuds du réseau (on l'appelle alors "racine") on possède une structure de contrôle de tous les noeuds du réseau à partir de cette racine. Si outre le fait d'avoir bâti une topologie d'arbre sur le réseau on peut s'assurer de la minimalité (à nombre de messages constant) de la somme des coûts unitaires de transition d'une arête, alors la structure de contrôle ainsi induite est optimale, à nombre de messages constant.

Pour le traitement informatique de ce problème, il y a alors deux façons de procéder :

a) On calcule la topologie d'arbre couvrant minimum à partir d'un site unique (calcul centralisé) doté d'une machine multiprocesseur type MIMD.

b) On utilise la structure en réseau pour implémenter un algorithme distribué basé sur la gestion des communications entre les processeurs.

Dans le premier cas, la construction de l'algorithme parallèle va dépendre de la structure de la machine parallèle, en particulier du mode de partage de la mémoire, du mode de partage des ressources en général (séquence exclusive).

Dans le second cas, la structure d'un algorithme distribué sera conditionnée par la logique inhérente aux communications entre les différents processeurs du réseau considéré. Communications avec rendez-vous, sans rendez-vous, avec ou sans "boite aux lettres", etc...

Pour ce qui nous concerne ici, nous allons nous intéresser au premier cas (pour le distribué, voir [LA,RO]).

1. Position du problème :

Mettons tout d'aobrd en évidence la propriété utilisée pour construire ces algorithmes.

1.1. Notations et définitions.

Considérons un graphe non orienté, valué défini comme suit :

$$G = (X,U,P)$$

où X est l'ensemble des sommets du graphe
U est l'ensemble des arêtes du graphe
P est une fonction de valuation de U dans $\mathbb{N}^*$.

L'application P induit un préodre sur U.

$$(\forall\, u_1, u_2 \in U\ ;\ P(u_1) \geq P(u_2)\ ) \Rightarrow (u_1 \geq u_2)$$

A - Arbre couvrant G

On dira qu'un arbre couvre G si tous les sommets de G sont sommets de l'arbre. On rappelle qu'un arbre couvrant G = (X,U) est une composante connexe de n-1 arêtes ($|X| = n$) et aussi qu'un arbre est sans cycle.

Notons A = (X,W) un arbre couvrant G où :

$$W \subseteq U$$

B - Poids d'un arbre

Etant donné un arbre A = (X,W), on appellera poids de l'arbre A et on notera $\mathbb{P}(A)$:

$$\mathbb{P}(A) = \sum_{(x,y)\in W} p(x,y).$$

1.2. L'algorithme de Sollin et Prim [BER], [FAU], [PRI].

Nous rappelons l'algorithme de Sollin tel qu'il est exposé dans [FAU], p.73, c'est à dire dans le cas séquentiel.

a) Distinguer les arêtes de même poids initial

b) A chaque étape de l'algorithme, choisir un sommet en dehors de ceux qui ont déjà été retenus dans la construction, et relier par l'arête de poids le plus faible, ce sommet à l'un des sommets auquel il est adjacent, l'arête ainsi sélectionnée et ses extrémités sont dits alors "retenus".

c) Lorsque l'ensemble des sommets a été utilisé entièrement (i.e. tous les sommets sont retenus) :
   - ou bien le résultat obtenu est un arbre, le problème est alors résolu
   - ou bien on n'a encore que plusieurs sous-arbres qu'on considèrera chacun comme l'un des sommets d'un multigraphe, les arêtes de ce multigraphe étant toutes les arêtes susceptibles de connecter deux à deux ces sous arbres ; on itère alors l'algorithme sur ce multigraphe.

1.3. Une propriété locale

Pour un graphe G = (X,U,P), connexe, on peut énoncer le théorème suivant :

Théorème

Pour tout sommet x, soit $\tilde{x}$ tel que :

$$p(x,\tilde{x}) = \min_{y \in \Gamma(x)} [p(x,y)],$$

il existe un arbre de poids minimal, couvrant le graphe G, et contenant l'arête $(x,\tilde{x})$. (G étant supposé connexe).

En effet, supposons qu'il existe un arbre couvrant G_1 ; $A_1 = (X, W_1)$ avec $(x,\tilde{x}) \notin W_1$; A_1 étant connexe, il existe un sommet t tel que $(x,t) \in W_1$; de plus :

$$p(x,t) \geq p(x,\tilde{x})$$

Dans A, il existe une chaîne et une seule entre x et $\tilde{x}$, soit $(x,\gamma),(\gamma,...),...,(...,\alpha),(\alpha,\tilde{x})$ cette chaîne. Or $p(x,\gamma) \geq p(x,\tilde{x})$. Par conséquent, si on supprime l'arête (x,γ) de A_1, on disconnecte A_1 en deux composantes connexes sans cycle : l'une C_1 contenant x, l'autre C_2 contenant $\tilde{x}$.

Par ajout de $(x,\tilde{x})$, on reconnecte C_1 et C_2, formant ainsi un nouvel arbre $A_2 = (X,W_2)$. Puisqu'on passe de W_1 à W_2 en substituant à l'arête (x,γ), l'arête $(x,\tilde{x})$, il vient :

$$P(A_1) \geq P(A_2)$$

2. PRINCIPE D'UN ALGORITHME PARALLELE POUR MULTIPROCESSEURS ASYNCHRONE [LAV1]

Pour construire un tel algorithme, on utilise la propriété locale énoncée précédemment. On peut énoncer la procédure générale suivante qui permet de bâtir les algorithmes que nous décrivons.

1) En tout sommet $x \in X$ (de façon asynchrone) associer $\tilde{x} \in X$ tel que $p(x,\tilde{x}) = \underset{y\in\Gamma(x)}{\text{Min}} [(x,y)]$, sans former de cycle

2) Construire le multigraphe $\overline{G} = (\overline{X},\overline{U})$ associé aux sous arbres.*)

3) Si $|\overline{X}| = 1$ Fin. (L'arbre minimal est alors constitué de l'ensemble des n-1 arêtes retenues en 1).
Sinon
recommencer avec $G := \overline{G}$.

2.1. Convergence de la méthode

A chaque "itération", $\overline{G}$ est constructible puisque G est connexe, d'autre part, le nombre de sommets (fini par hypothèse) de $\overline{G}$ diminue d'au moins une unité à chaque fois.

De plus, en vertu du théorème du paragraphe I.3, l'arbre ainsi trouvé est de poids minimal.

2.2. Les problèmes informatiques à résoudre

Le principal problème à résoudre est celui des conflits entre processus dans le cas où il y a égalité de poids sur certaines arêtes. Par exemple, considérons le graphe de la figure suivante :

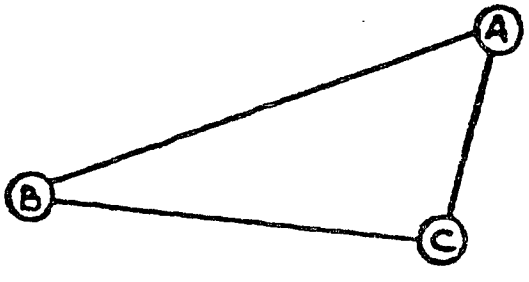


Figure 1

*) $\overline{G}$ est un multigraphe dont les sommets sont les sous-arbres de G et les arêtes, les arêtes de G dont les deux extrémités n'appartiennent pas à un même sous-arbre.

En appliquant la règle de sélection le processus associé au sommet A sélectionnera par exemple l'arête [A,B], celui associé à B l'arête [B,C], et celui associé à C, l'arête [C,A]. Ainsi on constituerait un cycle, ce qui est interdit.

Pour éviter ceci, il faut que lorsque deux processus A et B sont entrés en communication (i.e. l'arête [A,B] est introduite dans l'arbre), ni A, ni B, ni les ressources système nécessaires à leur communication, ne soient accessibles à un autre processus.

On peut schématiser comme suit le fonctionnement logique d'un processus :

1 - Identification

C'est une séquence du processus qui identifie le sommet du graphe initial auquel il faut connecter le processus (c'est à dire la composante connexe sans cycle - éventuellement réduite à un sommet - à laquelle est associé le processus en question)

2 - Sélection - Connexion

C'est une séquence qui sélectionne le processus qui gère le sommet précédemment identifié et qui connecte les processus entre eux. (C'est dans cette phase que se pose le problème du cyclage sus-mentionné, on utilisera alors la notion de "séquence exclusive").

3 - "Phagocytage", mise à jour

Lorsque le processus A entre en communication avec le processus B, A s'approprie toute l'information liée à B, relie les composantes connexes sans cycle associées respectivement à A et B, par l'arête qui a été sélectionnée en phase 1 et qui a servi à identifier le sommet géré par B.

Le processus A remet à jour les informations et variables globales affectées par ces modifications.

4 - Arrêt du processus, remise à disposition du système

Le processus B n'ayant plus aucun rôle à jouer peut s'arrêter. Les ressources système affectées au processus B (par exemple, un processeur) peuvent alors être remises à la disposition du système pour d'autres tâches.

On pourrait craindre un autre type de problème lié à la simultanéité d'action des processus. Ainsi A voulant entrer en communication avec B, B avec A et ceci simultanément. En fait, comme on s'est placé dans le cas asynchrone, on peut considérer que ce problème ne se pose pas. Dans le cas synchrone, il suffit alors de créer une relation d'ordre strict (une numérotation par exemple) sur les processus, et en cas de conflit, priorité est donnée au processus d'ordre supérieur.

2.3. Les moyens à mettre en oeuvre

On suppose qu'il y a un processus attaché à chaque sommet du graphe au départ soit, pour un graphe d'ordre n, n processus.

De plus, la gestion des variables globales et les communications entre processus se font par l'intermédiaire d'une mémoire centrale partageable. Les cellules de cette mémoire sont accessibles en lecture simultanément sans conflit. La mémoire centrale est accessible en écriture, mais de façon univoque.

Comme évoqué dans la phase 2, il y a des conflits à régler entre les processus pour éviter le cyclage, ceci s'obtient en interdisant pour certaines séquences de programme, le partage des ressources systèmes nécessaires à l'exécution de la dite séquence.

On considère donc ici qu'on dispose du point de vue logiciel, de la possibilité de rendre une séquence d'instructions exclusive (voir [RIC], [CAR]).

On entend par séquence exclusive, une séquence d'instructions appartenant à un processus telle que lorsque cette séquence est commencée, toutes les ressources (que ce soient des variables, des tableaux, ou des processeurs) susceptibles d'être utilisées par le processus actif au sein de cette séquence, soit attachées, durant toute l'exécution de la séquence (mais seulement de la séquence) au processus concerné de façon exclusive.

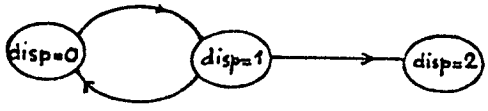
Afin de distinguer les variables locales propres à chaque processus, on conviendra de représenter tout identificateur de variable par un couple <N,C> où N est le n° du processus et C le nom de la variable (C pouvant être un tableau).

Par ailleurs, les variables globales (i.e. stockées en mémoire centrale partageable), partageables par plusieurs processus, seront désignées par un couple de la forme < 0,C >.

La notation <N,.> sera équivalente à <N,N>, c'est le numéro (l'identificateur) du processus.

2.4. Signification des différentes variables locales

< N,COMPT >	est un compteur qui contient le nombre d'arêtes du sous-arbre géré par le processus N.
< N,DISP >	est une variable d'état du procesus N.
DISP = 0	signifie que le processus N est entièrement disponible.
DISP = 1	signifie que le processus est non-disponible, mais qu'il peut le redevenir (il est en communication avec un autre processus). Le diagramme d'états d'un processus est donc le suivant :



DISP = 2	signifie que le processus N n'est plus disponible et ne le sera plus jamais, à sa prochaine activation le processus N se "suicidera" pour le problème en cours.
< 0,POINT >	est un tableau situé en zone globale de mémoire. Le tableau POINT permet de savoir quel processeur gère quel sommet {POINT T(I) = P} ⇔ {le processeur P gère le sommet I} POINT est accessible en lecture simultanément par plusieurs processus ; mais il n'est accessible en écriture que de façon exclusive.
≠	est le symbole de début ou de fin d'une séquence exclusive.
< N,ARBR >	est le tableau du sous-arbre géré par le processeur N.
∧ ∨	sont les opérateurs logiques "et" et "ou".
|	est l'opérateur de concaténation.
< N,LIST >	est un tableau propre au processeur N, il contient pour LIST(T) le numéro du sommet géré par N tel que < N,MAT(T) > donne la valuation de l'arête d'extrémité LIST(T) (dans N) et T (ailleurs, donné par <0,POINT(T)>).

2.5. La spécification de l'algorithme

(1) Tant que < N,COMPT > ≠ M - 1
/* On vérifie qu'on n'a pas déjà trouvé un arbre */
FAIRE :

(2) < N,T > ← (< N,T > : (< N,MAT(T) > = MIN(< N,MAT >)))
/* T est le nom du sommet relié par une arête de poids minimal au sous-arbre géré par N */

(3) ≢ (< N,DISP > = 0 ∧ < POINT(T),DISP > = 0) ∨ (< N,DISP > = 2)
/* on exécute la séquence exclusive si et seulement si cette expression logique est vraie, c'est une commande gardée */

FAIRE

SI < N,DISP > ≠ 2 ALORS < N,DSIP > ← 1 ; < POINT(T),DISP > ← 1
< N,Q > ← < 0,POINT(T) > ;
< 0,POINT(T) > ← < N,. >
/* Le processus dont le numéro est dans POINT(T) devient alors "esclave" du processus N lequel a alors accès à ses variables locales */

FSI
/* On donne la valeur 1 aux variables d'états des processus qui sont en communication et on remet à jour le tableau POINT dans lequel on écrit, et ce uniquement à travers cette séquence exclusive, on a auparavant sauvegardé le numéro du processus avec lequel on communique dans la variable locale < N,Q > */

FIN FAIRE
≢

(4) SI < N,DISP > = 2 then STOP FSI

(5) < N,COMPT > ← < N,COMPT > + < Q,COMPT > + 1 ;

(6) <N,ARBR> ← <N,ARBR> | <Q,ARBR> | {<N,LIST(T)>, <N,T>} ;
/* On remet à jour le compteur d'arêtes du processus, puis le tableau qui contient les dites arêtes (tableau ARBR). Les deux processus étant en communication, on considère que chacun d'entre eux a accès aux variables locales de l'autre */

(7) PROCEDURE A ;
/* voir ci-après */

(8) < Q,DISP > ← 2 ; /* Le processus Q ayant été "phagocyte", il est mis en "état terminal" */

(9) < N,DISP > ← 0 ; /* retour à l'état de disponibilité du processus N */

FIN FAIRE

(10) STOP

2.6. Fonctionnement de la commande gardée de la séquence exclusive

Après avoir dans l'instruction 2 identifié le sommet T candidat à la concaténation, il faut sélectionner le processus gérant T et entrer en communication avec lui. Ce processus est identifiable par POINT(T).

La communication entre les deux processus n'est possible que s'ils sont disponibles tous les deux, d'où la condition première de la commande gardée

$$[\ < N,DISP > = 0 \ \wedge \ < POINT(T),DISP > = 0] \ ....$$

Par ailleurs si on se contente de cette garde, il risque d'y avoir blocage si < N,DISP > = 2, puisque lorsqu'un processus entre dans l'état < N,DISP > = 2, il n'y a jamais retour à un état antérieur.

D'autre part, un processus n'est mis dans l'état < I,DISP > = 2 que par un autre processus et cet état peut apparaître à n'importe quel autre moment que pendant l'exécution de la séquence exclusive.

Donc à l'intérieur de cette séquence, on va tester cet état, et si < N,DISP > = 2, on passe l'exécution de la séquence exclusive et ce n'est qu'après qu'on pourra prendre la décision afférente à < N,DISP > = 2. Par conséquent, il faut autoriser un processus N à entrer dans sa séquence exclusive lorsque < N,DISP > = 2 d'où la commande gardée complète :

[(< N,DISP,ISP > 0 ∧ < POINT(T),DISP,ISP > 0) ∨ (< N,DISP,ISP > 2)]

De plus, la logique de la séquence exclusive veut que celle-ci ne soit exécutable que si la garde de la commande est vraie, or, si < N,DISP,ISP > 2 elle ne sera plus jamais vraie, et le processus Nss se mettra en attente indéfinie.

2.7.2 Fonctionnement de la procédure AE A

La procédure AE permet de transférer toute l'information liée au processus Qs changeant par Nr et de recalculer les poids des arêtes dont une extrémité (et une seule) est dans ce sous-arbre nouvellement formé, si deux telles arêtes ont même extrémité externe, on garde celle de plus petit poids qui est minimal.

```
PROCEDURE AE A
  DEBUT
    I ← 1
    TANTQUE I ≤ M
    FAIRE :
        SI (< Q,POINT(I) > = N ou < Q,POINT(I) > = Q)
          ALORS
        < N,MAT(I) > ← ∞ ,∞ ,
        < Q,POINT(I) > ← N
          SINON :
        SI (< N,MAT(I) > > < Q,MAT(I) >)
    FAIRE :
        < N,MAT(I) > ← < Q,MAT(I) >
        < N,LIST(I) > ← < Q,LIST(I) >
        fsi fsi
        fsi fsi
  FINPROC.
```

2.8.2. Evaluation des performances

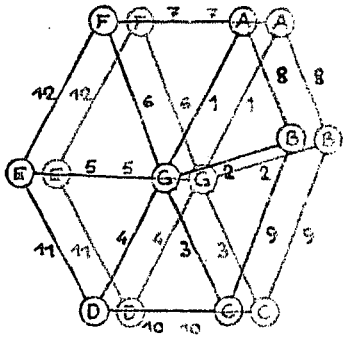
Il est très difficile de comparer les performances des différents algorithmes (ainsi, celui de [SAVA]) car ces hypothèses, tant sur le problème lui-même que sur le modèle de calcul, sont différentes. [SAVA] suppose par exemple que toutes les arêtes du graphe sont de poids distincts et qu'il dispose d'une machine SIMD avec n² processeurs.

Le fait de supposer qu'il n'y a pas deux arêtes de même poids élevé au carré, comme le fait remarquer [SAVA], le problème de conflit lié à la possibilité de cycle d'arêtes équipondérées, problème qui est résolu ici par la structure même de la séquence exclusive.

De plus, il existe un autre cas de conflit susceptible de dégrader considérablement les performances, c'est dans le cas où la structure du graphe infère qu'un seul processeur travaille à chaque étape.

On en a un exemple avec le graphe de la figure 2 bien qu'il n'y ait pas dans celui-ci deux arêtes ayant même poids.

Figure 2



Ainsi, dans le cas le plus défavorable, (diamètre du graphe égal à deux) il se peut qu'il n'y ait qu'un seul processus actif à chaque étape, par conséquent l'algorithme se déroule selon une procédure purement séquentielle. Il s'ensuit qu'il faut n-1 étapes (n-1 unités de temps) pour trouver le résultat. La complexité (en temps) de l'algorithme s'exprime en 0(n-1). Par contre dans le cas général, à chaque étape, chaque sommet se connecte à un autre, et, si on fait abstraction des temps de calcul de chaque processus pris individuellement (voir remarque), la complexité (en temps) s'exprime alors en $0(\lceil \log_2 n \rceil)$. Si on note α la complexité de l'algorithme, on a donc :

$$0(\lceil \log_2 n \rceil) \leq \alpha \leq 0(n-1)$$

Remarque

Dans le cas de machine MIMD, cette hypothèse de justifie par le fait qu'il y a recouvrement entre le temps de calcul de certains processus pris individuellement, et le temps de communication entre d'autres processus. De plus, le temps de calcul d'un processus n'est pas du même ordre de grandeur que celui des communications entre processus.

3. La programmation de cet algorithme

Le problème posé par la programmation de cet algorithme tient aux spécificités de la machine sur laquelle il doit être implémenté.

Ici, le programme est destiné à un CRAY-XMP4, ou à un CRAY 2.

Les n processus doivent donc se partager ici quatre processeurs et la gestion du parallélisme doit être assurée par le programmeur à travers les primitives disponibles dans le FORTRAN-CRAY.

On trouvera l'étude et la réalisation de cette programmation dans [FIL].

2. L'ENUMERATION IMPLICITE

Dans nombre de problèmes fortement combinatoires, c'est a dire ceux dans lesquels le nombre de solutions réalisables devient vite prohibitif, il s'agit d'éviter l'énumération exhaustive des solutions réalisables.

Pour ce faire, on a mis au point des méthodes d'énumeration implicite qui permettent d'évaluer si un ensemble de solutions est susceptible de contenir, ou non, une solution optimale. Il en est ainsi par exemple dans le problème de la minimisation des fonctions booléennes ou dans celui de la recherche d'un circuit hamiltonien de valeur minimale, ou, plus généralement dans la programmation linéaire en nombres entiers.

Les algorithmes d'énumeration implicite basés sur le principe des procédures arborescentes (P.S.E.S., P.S.E.P) sont les plus utilisées pour trouver des solutions optimales dans les problèmes combinatoires a forte complexité (problèmes dits N.P. complets).

Il en est ainsi des problèmes tels que l'ordonnancement des tâches, les problèmes dits de "voyageur de commerce" ou "des tournées", des problèmes d'affectation quadratique, ainsi que, plus généralement de la programmation mathématique en Nombres Entiers.

Si ces procédures permettent en général de limiter l'énumération, il s'avère malgré tout qu'il arrive un moment où le nombre de sommets pendants dans l'arborescence d'énumération des solutions devient si grand que l'exploration de ces différents sommets requiert un temps de calcul prohibitif. Ce caractère prohibitif du temps de calcul est en partie dû à l'aspect séquentiel de l'exploration. Un ordinateur séquentiel ne peut explorer qu'un seul sommet à la fois. Sur de telles machines, les problèmes de taille réelle ne peuvent souvent êtres résolus exactement à cause des dépassements de capacité mémoire (pour stocker le contexte de chaque sommet pendant), ou, le plus souvent à cause des dépassements du temps de calcul imparti qui provoquent l'arrêt du calcul avant l'obtention d'une solution optimale.

La parallèlisation de l'énumération implicite est donc une façon naturelle d'accélérer la recherche d'une solution optimale. Toutefois, cette parallélisation fait apparaître des phénomènes nouveaux inconnus en séquentiel.

Ces phénomènes sont pour l'essentiel les suivants:

- On peut être confronté à une dégradation des performances lorsque l'on augmente le nombre de processus. Et même, dans certains cas, on peut avoir des performances plus mauvaises en parallèle qu'en séquentiel si on ne prend pas garde.

- Suivant le compromis qu'on est amené à faire entre la distribution du travail aux processus, et le découpage de l'arborescence des solutions sous-traitée aux processus, on est amené à s'intéresser a la gestion des files d'attente des processus, de façon à ce qu'elles ne croissent pas indéfiniment, et au nombre de messages circulant sur le réseau d'interconnexion (ceci dans le cas distribué).

1. Le problème à résoudre.

Le problème combinatoire à résoudre est caractérisable de la façon suivante :

$$\min F(x),\ x \in S$$

où, S est un ensemble contenant l'ensemble des solutions réalisables, et où,

$$f : S \rightarrow R^{+}$$

est une fonction de coût; F(x) sera appelée valeur de la solution x.

De telles méthodes ont été généralisées et discutées par nombre d'auteurs [LAW], [MIT]; nous avons choisi ici de présenter brièvement le principe de ces méthodes (PSEP et PSES en français, Branch and Bound dans la littérature anglo américaine) comme dans [ROY].

Les procédures de recherche arborescente procèdent par énumération implicite des solutions du problème.

Elles sont construites sur trois notions clés :

- Un principe de séparation propre à tout noeud de l'arbre d'énumération associé qui sépare l'ensemble des solutions réalisables en une famille de sous-ensembles disjoints.

- Une borne inférieure: on associe à chaque sous-ensemble formé lors d'une séparation, une borne inférieure de la valeur de f, en résolvant par exemple, un sous problème relaxé associé.

- Une règle de sélection d'un sommet pendant de l'arborescence candidat à une nouvelle séparation, cette règle permet de se diriger dans l'arbre et de sélectionner les sous-ensembles de solutions sur lesquels il faut continuer à travailler.

Examinons plus en détail ces notions :

1.1. Principe de séparation.

L'ensemble des solutions réalisables est successivement divisé en sous ensembles dont les cardinaux sont de plus en plus petits (l'ensemble des solutions est évidemment discret et fini), de telle façon qu'il devient alors possible de résoudre le problème, ou du moins de déterminer s'il contient, ou non, potentiellement une solution optimale.

Le principe de séparation doit vérifier trois conditions:

1. La réunion de tous les sous-ensembles obtenus par séparation est identique à l'ensemble séparé.

2. Le nombre de séparations nécessaires pour atteindre une solution optimale est fini.

3. Lorsqu'un sous-ensemble de solutions ne peut plus être séparé, il est possible alors de déterminer si:

   - cet ensemble ne contient plus de solution réalisable, ou s'il est susceptible de contenir une solution meilleure que celle déja connue;

   - cet ensemble contient une solution réalisable; elle devient alors la meilleure trouvée.

Un tel sous-ensemble est appelé noeud terminal (ou feuille) par référence à l'arborescence. Ce noeud ne peut plus être séparé; le sous problème correspondant est abandonné.

1.2. Le principe d'évaluation.

Aprés avoir séparé un ensemble de solutions en sous-ensembles, il est nécessaire de calculer une borne inférieure des solutions pour chaque sous-ensemble formé.

Ainsi, on définit la fonction "valuation"

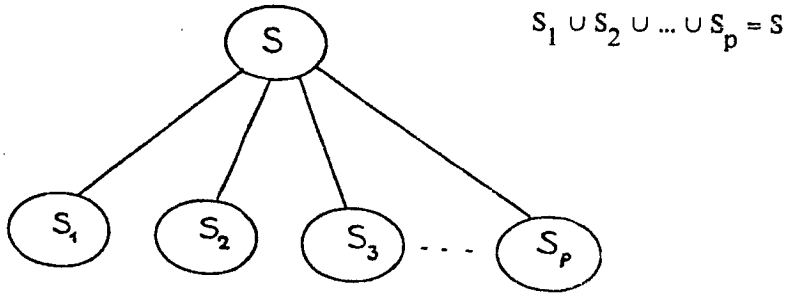
$$\nu : P(V) \rightarrow R^{+} \ (U \in V, V \text{ ensemble de solutions})$$

telle que :

- Si S est un sous ensemble de solutions, alors $\nu(S) \leq \min_{(x \in S)} f(x)$.
- Si S est réduit à une seule solution réalisable, alors $S = \{x\}$ et $\nu(S) = f(x)$.

Une telle procédure peut être illustrée par une arborescence, dont chaque noeud correspond à un sous-ensemble de solutions, les successeurs d'un noeud correspondant aux sous-ensembles créés lors de la séparation (voir figure 1.) .

Figure 3



1.3. Stratégies de recherche

C'est la règle de sélection qui permet de savoir sur quel ensemble continuer à travailler, c'est à dire comment pratiquer la séparation, ou, le cas échéant, l'abandon pur et simple du sommet considéré.

En général, on utilise différentes règles afin de réduire la taille finale de l'ensemble des solutions réalisables.

2.4.1. En profondeur d'abord

Le noeud suivant qu'on examine est l'un des successeurs direct du noeud courant (on dit que c'est l'un [illegible] fils). Cette stratégie a l'avantage de limiter la taille de l'espace mémoire nécessaire à sa

mise en oeuvre, et elle permet de trouver rapidement une solution réalisable. L'avantage procuré par le fait de trouver rapidement une solution réalisable est dû à deux choses :

i) La valeur de cette solution réalisable fournit une borne **maximale** des valeurs d'une solution ; d'où par la suite la possibilité d'éliminer des branches de l'arbre, c'est à dire encore de limiter l'énumération. on procède ainsi à un encadrement de la valeur optimale qui se resserre au fur et à mesure qu'on exhibe des solutions réalisables d'une part, qu'on fait croître la borne inférieure de la valeur prise par la fonction d'évaluation d'autre part.

ii) au cas où le temps de calcul deviendrait prohibitif et qu'il faudrait alors arrêter le calcul, on possèderait au moins une solution réalisable.

1.3.2. En largeur d'abord

La règle associée est alors de continuer la séparation sur le noeud correspondant à la meilleure borne inférieure trouvée (c'est à dire à celle de valeur maximale). Cette stratégie est comme dans la littérature anglo-saxonne sous le terme Branch and Bound (B&B).

1.3.3. Stratégie mixte

Il s'agit d'une combinaison des deux stratégies précédentes. Cette façon de faire s'apparente d'une certaine manière à la procédure dite alpha/béta, telle qu'elle est pratiquée en intelligence artificielle.

1.4. La polychotomie

Supposons connues pour chaque noeud de l'arbre d'énumération les propriétés caractéristiques d'une solution réalisable ou d'une solution non réalisable obtenue par relaxation de certaines des contraintes du problème initial.

Pour simplifier, explicitons le mode de séparation du noeud racine S_0, ensemble des solutions réalisables du problème.

Admettons que la solution réalisable doive avoir les propriétés $P_1, P_2, \ldots, P_k$. Pour $i = 1, 2, \ldots, k$, S_i note l'ensemble des solutions qui ont la propriété P_i, $\bar{S}_i$ l'ensemble complémentaire de S_i (par rapport à S_o).

Une séparation de S_0 est une partition en $k + 1$ sous-ensembles avec :

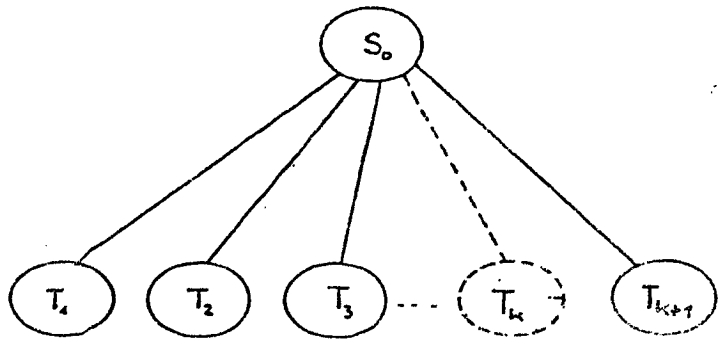
$$T_1 = \bar{S}_1,\ T_2 = \bar{S}_2 \cap S_1,\ T_3 = \bar{S}_3 \cap S_2 \cap S_1$$

$$T_k = \bar{S}_k \cap S_{k-1} \cap \ldots \cap S_1,\ T_{k+1} = S_{k+1} \cap S_k \cap \ldots \cap S_1\ ,\ \text{avec :}$$

$$\bigcup_{i=1}^{i=k+1} T_i = S_o \text{ et } T_i \subseteq S_i$$

En fait, seuls les noeuds successeurs, $T_1, \ldots, T_k$ doivent être créés ainsi que doit être calculée une borne inférieure des solutions afférentes.

Figure 4



un niveau de l'arbre d'énumération polychotomique.

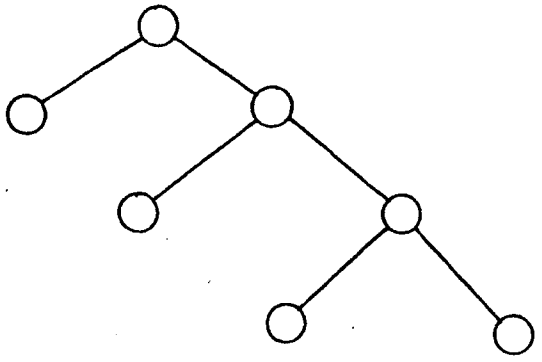
Le noeud T_{k+1} est un noeud terminal, il correspond à une solution réalisable ayant les propriétés $P_1, P_2, \ldots, P_k$; il n'est pas nécessaire de créer un tel noeud.

Il est évident que la partition dépend de l'ordre dans lequel on a indexé les propriétés $P_1, \ldots, P_k$.

Dans le principe de séparation dichotomique, chaque noeud n'est successivement séparé qu'en deux autres noeuds à chaque niveau de l'arbre.

Ainsi, avec une stratégie "en profondeur d'abord", l'arbre dichotomique correspondant aux séparations de la figure 1 serait celui de la figure 5.

Figure 5



k niveaux de l'arbre d'énumération dichotomique

2. Stratégies de parallélisation

Il y a au moins deux façons d'aborder le problème de la parallèlisation d'une méthode d'exploration arborescente:

2.1. En minimisant l'énumération.

On veut éviter toute énumération inutile de sommets de l'arborescence. C'est à dire qu'on veut d'une part éviter la redondance, et d'autre part éviter de continuer à séparer des sommets qui ne peuvent conduire à des solutions optimales.

Ces deux cas peuvent se produire si:

i) La façon d'affecter un sommet à séparer à un processus n'est pas univoque (risque alors de redondance comme dans [DE,HU]).

ii) Un processus continue d'énumérer des sommets pour lesquels la valeur prise par la fonction d'évaluation est moins bonne que celle trouvée dans une autre branche de l'arbre, par un autre processus.

Donc, pour éviter ces deux écueils, il faut:

a) Une manière univoque d'affecter un processus à un sommet de l'arbre d'énumération.

b) Un moyen de mettre à jour, chaque fois qu'elle change, la meilleure borne possible pour tous les processus.

2.2. En minimisant les communications.

On veut alors limiter les communications entre processus afin de ne pas trop charger le réseau de communications du système.

Le temps de communication entre deux processus est en général plus long que le temps nécessaire à une étape élémentaire de calcul d'un processus. Dans le cas où il en est ainsi, on peut être tenté de penser qu'en réduisant les communications entre les processus, on réduit le temps global de calcul. Ce raisonnement toutefois, ne tiens pas compte de "l'explosion" combinatoire du problème. En effet, des processus non communicants entre eux vont énumérer beaucoup. Le gain de temps lié aux communications économisées sera vite compensé par le nombre de sommets de l'arborescence, inutilement énumérés.

On va donc, contrairement à [DE,HU] et [MOH.], s'attacher à élaborer des algorithmes les moins énumératifs possible.

L'avantage de la séparation polychotomique sur la séparation dichotomique tient au fait qu'on a une information plus précise (bornes plus fines) en chaque noeud pour choisir celui qui doit être séparé a l'étape suivante.

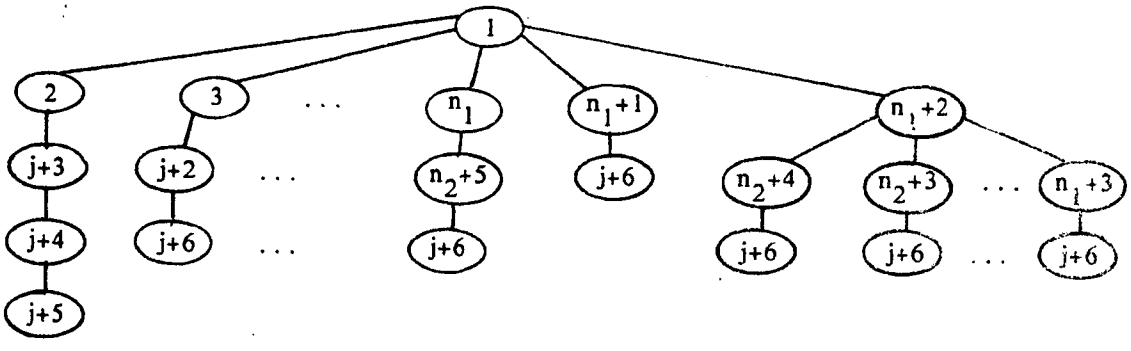
De plus, si on utilise une dichotomie, le nombre de sommets créés pour l'arborescence, pour le même nombre de sommets pendants, est plus important et par conséquent, le nombre de messages émis est beaucoup plus grand que dans le cas d'une polychotomie.

La polychotomie permet une gestion dynamique; à chaque étape la valeur de k dépend des propriétés de la solution réalisable connue. En [ROU.], il a été montré que l'utilisation de tels scénarios de branchement est trés efficace dans beaucoup de problèmes combinatoires classiques tels que le problème du voyageur de commerce [BEL], la programmation linéaire en nombres entiers ou en variables bivalentes [GAR], le problème de l'affectation quadratique. De plus, la polychotomie est particulièrement bien adaptée pour le parallèlisme asynchrone.

3. Anomalies dans la parallélisation des méthodes arborescentes [LA,SA]

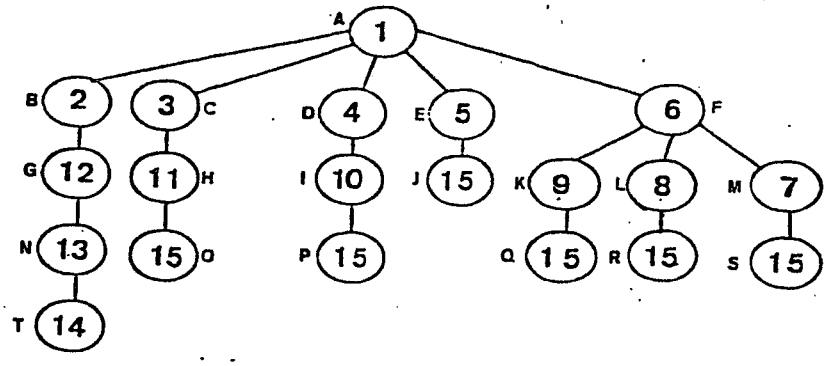
Dans un algorithme séquentiel d'une procédure SEP, chaque itération consiste à appliquer le principe de séparation sur le sommet pendant de meilleure évaluation. Si l'on dispose de plusieurs processeurs, on pourra alors, à chaque itération, examiner en parallèle plusieurs sommets de l'arborescence.

A priori pour un problème, le fait d'augmenter le nombre de processeurs devrait conduire à une diminution du nombre d'itérations. Ce résultat est souvent vérifié, mais on construit facilement des contre-exemples. Ainsi il est possible de trouver des problèmes tels que, le passage d'un nombre n_1 de processeurs à un nombre n_2 s'accompagne d'un accroissement du nombre d'itérations. Il suffit pour cela de considérer le problème représenté par l'arborescence suivante :



avec $j=n_1+n_2$

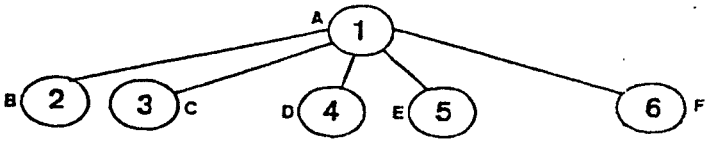
Considérons par exemple le cas où $n_1 = 4$ et $n_2 = 5$. L'arborescence est alors la suivante :



La solution optimale correspond au sommet terminal T.

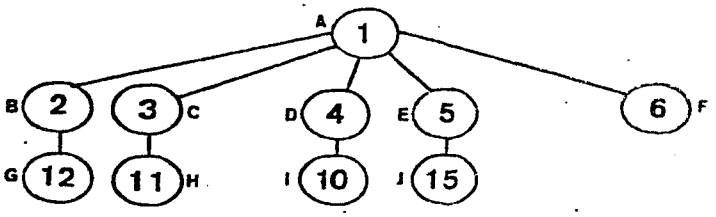
Supposons que l'on dispose de 4 processeurs. A chaque itération, 4 sommets au plus pourront être examinés, choisis dans l'ordre de leur évaluation. La construction de l'arborescence se fait selon les étapes suivantes : (On fait une hypothèse implicite de fonctionnement synchrone, en traitant l'arborescence niveau par niveau).

1ère itération



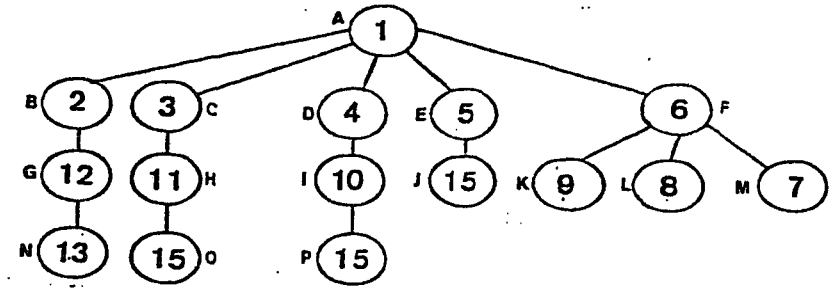
Examen du sommet A

2ème itération



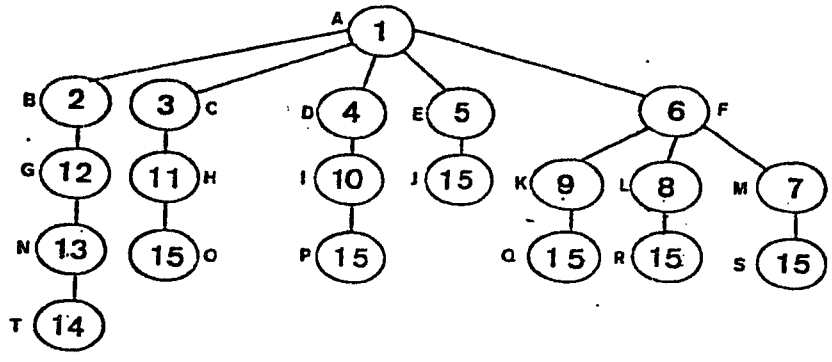
Examen des sommets B, C, D, E

3ème itération



Examen des sommets F, G, H, I

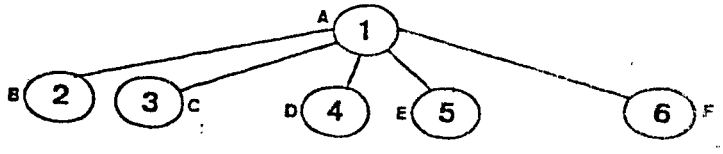
4ème itération



Examen des sommets K, L, M, N

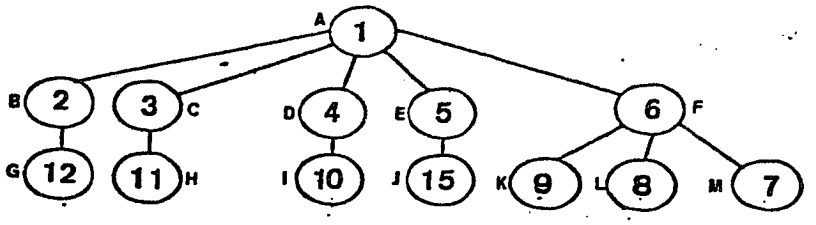
On obtient donc la solution optimale au bout de 4 itérations. Supposons maintenant que l'on dispose de 5 processeurs. La procédure se déroule alors de la manière suivante :

1ère itération



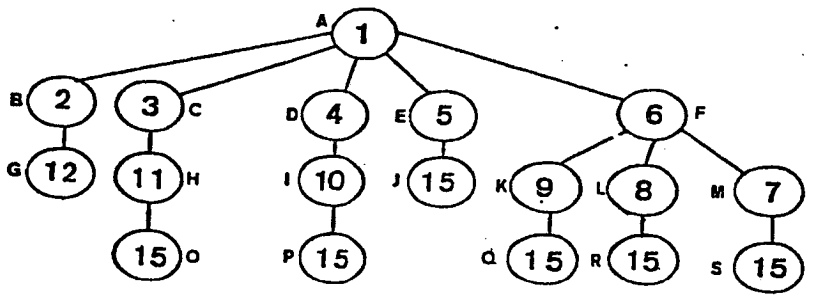
Examen du sommet A

2ème itération



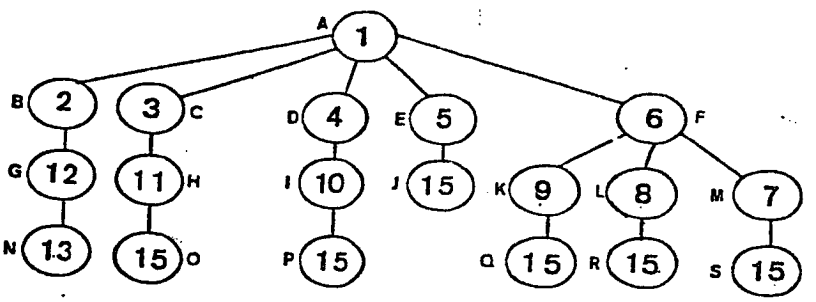
Examen des sommets B, C, D, E, F

3ème itération



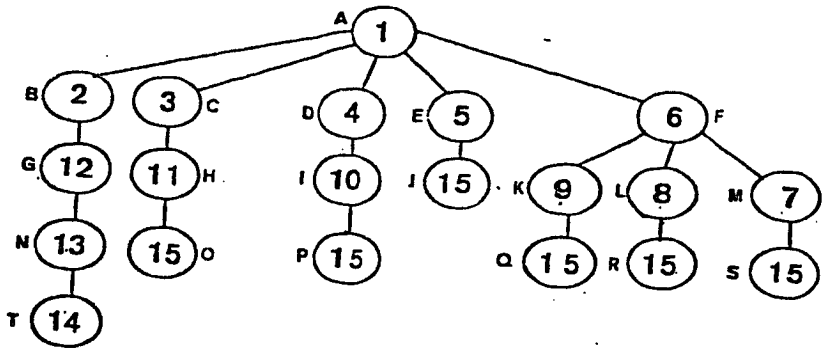
Examen des sommets H, I, K, L, M

4ème itération



Examen du sommet G

5ème itération



Examen du sommet N

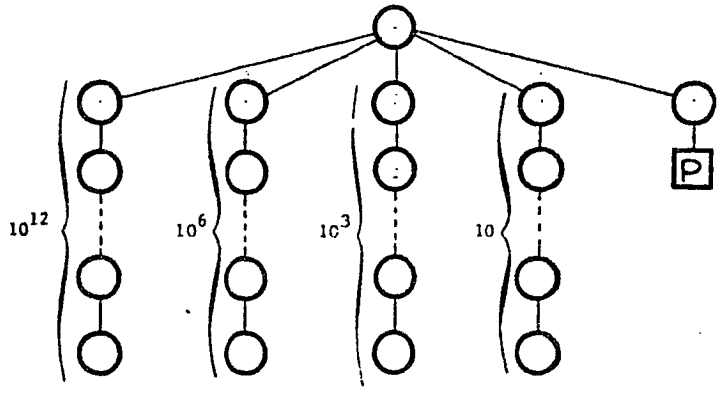
Il a fallu exécuter une itération supplémentaire alors que l'on disposait d'un processeur de plus. On met ainsi en évidence une instance pour laquelle l'accroissement du nombre de processeurs n'entraine pas une diminution du nombre d'itérations, mais au contraire une augmentation de celui-ci.

4. Sur-accélérations

Inversement, on peut mettre en évidence des anomalies favorables à la parallélisation. En effet, on pourrait s'attendre à ce que le nombre d'itérations diminue, dans le meilleur des cas, proportionnellement à l'accroissement du nombre de processeurs, soit :

$$\frac{I(n_1)}{I(n_2)} \leq \frac{n_2}{n_1}$$

Toutefois on peut mettre en évidence des instances de calcul pour lesquelles cette relation n'est pas vérifiée. Considérons par exemple l'arborescence suivante des solutions d'un problème.



Supposons qu'on adopte comme stratégie de recherche un "calage à gauche" tout le temps, indépendamment du nombre de processus disponibles.

Alors sur un tel problème, ce qu'il est convenu d'appeler la "thèse élargie sur le parallélisme" est prise en défaut.

Plaçons nous à chaque fois dans le cas le plus défavorable comme il est d'usage en analyse de complexité ; c'est-à-dire ici, supposons que la solution cherchée soit au sommet P.

La stratégie de parcours de l'arborescence employée sera ici un back track à gauche.

Nous allons estimer les accélérations obtenues avec 2, 3 4 et 5 processeurs.

Avec un processeur, le nombre d'itérations est

$$t_1 = 10^{12} + 10^6 + 10^3 + 10 + 2 ,$$

avec deux processeurs,

$$t_2 = 10^6 + 10^3 + 10 + 2 ,$$

avec trois processeurs,

$$t_3 = 10^3 + 10 + 2,$$

avec quatre processeurs,

$$t_4 = 10 + 2 ,$$

et avec cinqu processeurs,

$$t_5 = 2.$$

Ce qui induit des accélérations bien supérieures à ce que laisse prévoir l'inégalité de la première page. Ici, on a :

pour deux processeurs $\frac{t_1}{t_2} \approx \frac{10^{12}}{10^6} >> 2 ,$

pour trois processeurs $\frac{t_1}{t_3} \approx \frac{10^{12}}{10^3} >> 3 ,$

pour quatre processeurs $\frac{t_1}{t_4} \approx \frac{10^{12}}{10} >> 4 ,$

et enfin avec cinq processeurs $\frac{t_1}{t_5} \approx \frac{10^{12}}{2} >> 5$

ce qui est en contradiction flagrante avec l'inégalité pré-citée.

5. Retour au calcul séquentiel

Examinons la contradiction évoquée ci-dessus. En fait cette contradiction a été mise en évidence pour des instances particulières d'un problème donné. La question se pose alors de savoir s'il n'existe pas un algorithme séquentiel dont les performances soient telles que l'accélération engendrée par le

passage au parallélisme soit toujours inférieure ou égale au nombre de processeurs utilisés en parallèle. C'est-à-dire qu'avec q processeurs on ait toujours :

$$\frac{t_1}{t_q} \leq q \quad \left| \begin{array}{l} t_1 \text{ nombre d'itérations avec 1 processeur} \\ t_q \text{ nombre d'itérations avec q processeurs.} \end{array} \right.$$

On peut effectivement construire un tel algorithme séquentiel en introduisant la notion de processeurs.

On crée cinq processus, chacun d'entre eux étant associé à l'un des sommets de premier niveau de l'arbre.

Séquentiellement, on exécute un pas de calcul du premier processus, puis à l'étape suivante un pas de calcul du deuxième processus, ainsi de suite jusqu'au cinquième. De cette façon, on aura séquentiellement "simulé" le parallélisme et le nombre de pas de calcul effectués par un processus sera le même en séquentiel et en parallèle.

Compte non tenu des pertes de temps calcul inhérentes à la machine, l'inégalité $\frac{t_1}{t_q} \leq q$ sera bien vérifiée.

6. Conclusion

Nous avons vu comment d'un algorithme séquentiel, par simple généralisation, on pouvait déduire un algorithme parallèle. La mise en évidence d'"anomalies" et de sur-accélérations nous a conduit à construire un nouvel algorithme pour le cas séquentiel qui est en fait une simulation du parallélisme par une machine séquentielle.

On peut se demander si avec les possibilités du parallélisme on ne peut pas trouver des algorithmes qui soient spécifiques. Auparavant, il nous faut faire une remarque sur l'étude faite aux §. 3, 4 et 5. En effet, dans ces paragraphes, afin de visualiser des instances de calcul, nous avons fait une hypothèse implicite de synchronicité des calculs. En fait, il faut raisonner en termes asynchrones. Dans un tel cas, l'algorithme décrit au §. 5 perd de sa pertinence en ce qui concerne la réflexion sur les coefficients d'accélération.

Dans une machine parallèle, le fait de considérer que les différents processeurs fonctionnent de façon asynchrone renvoie, pour le modèle théorique afférent au non-déterminisme, et à un modèle théorique d'algorithme parallèle (sur ce sujet, voir [LAV4] ou [LA,LA]).

Par conséquent, construire des algorithmes spécifiques au parallélisme asynchrone passé par la prise en compte de ce non-déterminisme. Par exemple, on peut lancer un certain nombre de processus dans une recherche du type "en largeur d'abord", et d'autres dans une recherche "en profondeur d'abord", et ceci simultanément.

Mais là encore, même si l'aspect asynchrone interdit une simulation séquentielle pratique de toutes les instances possibles de calcul (à cause de la combinatoire qui en résulte), on est encore d'une certaine façon dans une vision généralisée d'un algorithme séquentiel.

Une voie de recherche intéressante serait certainement de concevoir un algorithme parallèle dans lequel les processus adopteraient un comportement "intelligent", choisissant en fonction de règles une recherche en profondeur ou en largeur.

Une autre voie de recherche théorique certainement prometteuse serait aussi de raisonner en termes d'algorithmes non-déterministes et d'utiliser le parallélisme pour ce faire.

Note : Nous avons conçu un algorithme parallèle de recherche arborescente pour le problème de l'affectation quadratique. On en trouvera la spécification et l'implémentation sur CRAY- XMP4 dans [LA,VE].

BIBLIOGRAPHIE

[BEL] BELLMORE M. and MALONE J.C., "Pathology of Traveling Salesman Subtour Elimination Algorithms", Operations Research 19, 278-307, 1971.

[BEN] BENTLEY J.L., "A parallel algorithm for constructing minimum spaning trees", Journal of algorithms 1, 51-59, 1980.

[BER] BERGE C., "Graphes & hypergraphes", Dunod 1976.

[BRE] BRENT R.P., "The parallel evaluation of general arithmetic expressions", J.ACM 21, 2 (1974), 201-206.

[BU,K,R,S] BURTON F.W., McKEOWN G.P., RAYWARD-SMITH V.J., SLEEP M.R., "Parallel Processing and Combinatorial Optimization", School of Computing Studies and Accountancy, University of East Anglia 1984, Norwich NR4 7IJ, UK.

[CAR] CARVALHO Q.S.F., ROUCAIROL G., "On mutual exclusion in Computer networks", Technical correspondance, CACM, vol 26, n°2, Feb. 1983.

[CH,TA] CHERITON D., TARJAN R.E., "Finding Minimum Spanning trees", J. SIAM Comp., 5, 1976, pp. 724-742.

[CH,LA,CH] CHIN F.Y., LAM J., CHEN N.I., "Efficient Parallel Algorithms for some Graph Problems", CACM, Vol 25, n° 9, pp. 659-665.

[COR] CORNELUS Ph., "Algorithmes Parallèles de Cheminement dans les graphes" Mémoire d'Ingénieur IIE, Juin 1985. CNAM, 292, rue St Martin, 75003 Paris, France.

[DE,PA,LO] DEO N., PANG C.Y., LORD R.E., "Two parallel algorithms for shortest path problems", Proc. 1980 Internat. Conf. Parallel Processing, 244-253.

[DE,HU] DESSOUKI O.I., HUEN W.H., "Distributed Enumeration on Network Computers", IEEE Trans. on Comp., Vol C-29, n° 9, Sept. 1980.

[DIJ] DIJKSTRA E.W., "A Note on Two problems in connection with graphs", Numerische Mathematik 1, 1959, pp. 269-271.

[FAU] FAURE R., "Précis de recherche opérationnelle", Dunod 1971.

[FEI] FEILMEIR "Parallel Numericals Algorithms" in Parallel Processing Systems, Ed. David J. Evans. Cambridge University Press 1982.

[FIL] FILALI L., "Arbre couvrant et parallélisme", Mémoire d'ingénieur IIE, CNAM Sept. 1986.

[FOU] FOURNIER J.C., "Introduction à la notion de matroïde", Publications mathématiques d'Orsay 79-03, Université Paris Sud, 91405 Orsay, France.

[FO,L,R,S] FOX B.L., LENSTRA J.K., RINNOOY KAN A.H.G., SCHRAGE L.E., "Branching from the largest upper bound. Folklore and facts", European Journal of Operations Research 2, 1978, pp. 191-194.

[FRA] FRAISSE P., "Vectorisation d'algorithmes de théorie des graphes", Mémoire de DEA, 25 juin 1984, LRI, Bât. 490, Université Paris Sud, 91405 Orsay Cédex

[GAR] GARFINKEL R.S., "On partitionning the feasible set in branch and bound algorithm for the asymmetric traveling salesman problem", Operations researches 21, 340-343, 1973.

[HO,MA,SI] HOCHSCHILD P.H., MAYER E.W., SIEGEL A.R., "Techniques for solving graph problems in parallel environments", IEEE, 1983.

[KA] KARNIN E.D., "A parallel algorithm for the Knapsack problem", IEEE Trans. on Comp., vol C-33, n° 5, May 1984.

[KE,RA] McKEOWN G.P., RAYWARD-SMITH V.J., "Chaotic Computing", Internal report CSA/15/1984, University of East Anglia, Norwich NR4 7TJ, UK.

[KNU] KNUTH D.E., "The art of computer programming : fundamental algorithms", Addison-Wesley Publishing Company, 1973.

[KRU] KRUSKAL J.B., "On the shortest spanning subtree of a graph and the traveling salesman problem", Proc. Amer. Math. Society, 7, 1956, pp. 48-50.

[LA,LA] LAVALLEE I., LAVAULT C., "Algorithmique parallèle et distribuée", Rapport de recherche INRIA n° 471, Dec. 1985.

[LA,RO] LAVALLEE I., ROUCAIROL G., "A fully distributed (minimum) spanning tree algorithm", Information processing letters, Août 1986, pp. 55-62.

[LA,RO1] LAVALLEE I., ROUCAIROL C., "A parallel Branch and Bound algorithm", Rapport de recherche n° 164, LRI, Bât. 490, Université Paris Sud, 91405 Orsay (France), 1984.

[LA,SA] LAI T.H., SAHNI S., "Anomalies in Parallel Branch and Bound Algorithms", CACM, Vol. 27, n° 6, juin 1984.

[LA,VE] LAURENT P., VERLEYE C., "Procédures arborescentes sur machines parallèles et sur réseaux de processeurs", Mémoire d'ingéneiur IIE, 1985, CNAM, 292, rue St Martin, 75003 Paris, France.

[LAV1] LAVALLEE I., "Notes sur le parallélisme", Rapport interne GR22, Université Paris VI, Avril 1983.

[LAV2] LAVALLEE I., "An efficient parallel algorithm for computing a minimum spanning tree", Parallel Computing 83, Elsevier Science Publishers B.V. (North Holland), pp. 259-262, 1984.

[LAV3] LAVALLEE I., "Un algorithme parallèle efficace pour construire un arbre couvrant minimal dans un graphe" RAIRO série verte, Fev. 1985.

[LAV4] LAVALLEE I., "Contribution à l'algorithmique parallèle et distribuée, application à l'optimisation combinatoire", Thèse de doctorat d'état es sciences. LRI Université Paris XI, Bât. 490, 91405 Orsay Cédex, juin 1986.

[LAW] LAWLER E.L. and WOOD D.E., "Branch and Bound Methods : a survey", Operations research 14, 699-719, 1984.

[LE,KI] LENSTRA J.K., KINDERWATER G.A.P., "Parallel Algorithms in Combinatorial Optimisation, an annotated bibliography", M.C. Amsterdam 1983.

[MAF] MAFFIOLI F., "Complexity of optimum undirected tree problems", Analysis and Design of algorithms in Combinatorial Optimization, Edited by Ausiello G. Lucertini M. Springer Verlag 1981.

[MIT] MITTEN L.G., "Branch and Bound Methods : General Formulation and Properties", Operations Research 18, 24-34, 1970.

[MOH] MOHAN J., "A study in parallel computation : the travelling salesman problem", Department of Computer Science, Carnegie Mellon University, CMU-CS-82-132, August 1982.

[PRI] PRIM R.C., "Shortest connections networks and some generalizations", Bell System Tech. J., Vol. 36, 1957, pp. 1389-1401.

[QU,DE] QUINN M.J., DEO N., "Parallel graph algorithms", Comp. Survey, vol. 16, n° 3, Sept 1984.

[RIC] RICART G., and AGRAWALA A., "An optimal algorithm for mutual exclusion in computer networks", Comm. ACM 24-1, Jan. 1981., pp. 9-17.

[RO] ROUCAIROL C., "A parallel Branch and Bound Algorithm for the quadratic assignment problem", Rapport de recherche, MASI, 4 place Jussieu, 75230 Paris cédex 05 (France), 1985.

[ROU] ROUCAIROL C., "Un principe de séparation efficace dans les méthodes d'exploration arborescentes : polychotomie ou partition en k sous-ensembles (k>2)", Rapport interne Institut de programmation, Université Paris VI, Mai 1983.

[ROY] ROY B., "Procédures d'exploration par séparation et évaluation", RAIRO verte, 3ème année, Vol 1, pp. 61-90, 1969.

[ROY] ROY B., BENAYOUN R., TERGNY J., "From S.E.P. Procedure to the Mixed Ophelie program", Integer and non linear programming, ed. J. Abadie, North-Holland, 1970.

[SAV] SAVAGE C., JA'JA J., "Fast efficient parallel algorithmes for some graph problems", SIAM J. on computing, vol 10, Nov. 1981, pp. 682-691.

[SOL] SOLLIN M., Exposé du séminaire de C. Berge, IH.P., 1961, repris in extenso dans "Méthodes et modèles de la R.O", Tome 2,pp. 33-45, A. Kaufmann, Dunod 1968.

[YAO] YAO C.C., "An O(| E | loglog | V |) algorithm for finding minimum spanning trees", Inf. Proc. Letters, 4, 1975, pp. 21-25.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique