Algorithmique parallele et distribuee

Ivan Lavallee, C. Lavault

INRIA
CENTRE DE 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° 471

ALGORITHMIQUE PARALLÈLE ET DISTRIBUÉE

Ivan LAVALLÉE
Christian LAVAULT

Décembre 1985

ALGORITHMIQUE PARALLELE ET DISTRIBUEE

Ivan LAVALLEE
Christian LAVAULT

ABSTRACT

The parallel execution time has been studied extensively during the past ten years, primarily with the goal of finding sole processing activies time costs of parallel algorithms on several kinds of PRAM's models. On the other hand, a sheer distributed approach proved a fruitful strand of research for communication activities measures. The first part is devoted to parallel and distributed algorithms in general. A complete definition of paralleland distributed algorithms is given. Taking both aspects of the problem into account, we propose an overall "efficiency-performance measure" of parallel and distributed algorithms. Focusing on this synthetic point of vue, wethen define a general abstract model of distributed algorithm for measure of "efficiency-performance" which is a very complete one. In the second part, three fundamental algorithms are studied in the worst-case, with help of a weakened version of the latter model : Maximum-finding, Merging and Sorting algorithms of Valiant. The implementation on the model's framework achieves Valiant's and Kruskal's worst-case optimal bounds whenever the number of processes is smaller that the size of the problem.

RESUME

Ce rapport est divisé en deux parties. Dans la première partie, nous donnons une définition des notions d'algorithmes parallèles ou distribués qui nous permet d'aborder de façon synthétique les notions de complexité, d'efficacité etc... de ces algorithmes. Nous montrons comment prendre en compte à la fois les aspects de la complexité sur des modèles type PRAM et sur des modèles du type distribué. Dans la deuxième partie, nous appliquons le modèle précédent à trois algorithmes fondamentaux qui sont : la recherche du maximum, le tri et la fusion. Nous montrons alors à partir de notre modèle comment évolue la complexité des algorithmes de Valiant lorsqu'on tient compte des problèmes de communication.

ALGORITHMIQUE PARALLELE et DISTRIBUEE

- **Premiere partie:**

CONCEPTS DE BASE, COMPLEXITES, EFFICACITE.

- **Deuxieme partie:**

APPLICATION A TROIS PROBLEMES CLASSIQUES

LAVALLEE Ivan

LAVAULT Christian

- EXTENDED ABSTRACT -

INTRODUCTION

The emergence of parallel computation in computer science gives a theoric challenge needing new theoric answers in the design and analysis of algorithms. It is actually the very concept of algorithm itself which must be completely rethought.

Since the classical model of Turing machine upon which algorithm's notion is founded is essentially serial, a general and abstract model for distributed or parallel algorithms and their complexity measures is needed. Each of the models recently proposed (P-RAM, P-RAC, W-RAM, of all kind, see [BO, HO.], uni- and bi-directionnel rings, trees and pyramid machines, etc...) is actually taking into account only one aspect of the complexity problem of parallel algorithms at a time. For example, the synchronisation between processes in the actual shared memory system of P-RAM models tends to be based upon *shared* variables and memory. However, a much more natural approach can be settled with distributed systems and processes message passing through a communication medium (e.g., a bus or a communication network. See [SAN.] for distributed networks model).

As a matter of fact, one has to reckon with future (or even already actual) abilities of large distributed multiprocessors parallel machines. Besides, a theorical and abstract *model for complexity measures* of parallel and distributed algorithms must stand free from the proper constraints of machines' architecture *and* , at the same time, must come to appear more complementary to real implementable multiprocessors computers.

Thereby, such a model would entail at any rate three major qualities : first, be released enough from "fetch-and-add" memory conflicts and from any "architecture dependancy"; second, be "robust" with regard to fundamental highly efficient parallel algorithms; and third, be general and adaptable enough for parallel as well as distributed computation. In this respect, the

theorical model proposed in this publication seems indeed rather instrumental and convenient as well as realistic and general.

The abstract model specified below enables to introduce a "new" definitions of algorithmic complexity measures which estimates the "performance" of parallel or distributed algorithms. This new notion of *performance measures* must be prefered, in our opinion, for better faithfulness to the reality of parallel and distributed computing. Two difficulties indeed are to be avoided in the matter.

First, the setting of a theory which would appear to be deeply disconnected with the reality of effective computing - that is to say the mediatization of algorithm's *execution* by a given machine.

Second, the building of a too "architecture dependant" model.

THE MODEL

We define a distributed algorithm (parallel algorithm can be steadily derived from the latter) as an octuplet:

$$A = (P, G, A, S_L, S_F, E, Q, D)$$ [1]

Where:

1) $P = \{ p_1, \cdots, p_n \}$ is the finite set of necessary processes .

a) We define a process as a serial algorithm with communication capabilities.

b) Each process has his own "waiting room" (buffer) : a discrete-time queuing M/M/1/c/FIFO system (exponentially distributed service times and Poisson arrivals).

2) $G = (P, L)$ is an undirected graph with no self-loops or multiple edges. It is the potential/possible graph of communications between processes. Vertices are processes and edges the set L of possible direct connections from one process to another (or fixed communication channels linking them). At the same time, G settles a closed markovian M/M/n/n.c/FIFO queuing network (where n.c is the maximum capacity of the network). E denotes the set of states of the Markov queuing network (these states are time-dependant).

[1] See [SAN.] and [RO,SA,SI.] for fundamental, though essentially focused on message complexity, characterization and analysis of distributed algorithms.

This markov model is a classical renewal model with equilibrium joint queue distribution of "product form" (it is often called a "BCMP-type" network. See [GE,PU.], [LO,LA.]). The queuing network model mapped on to the communication graph G has properties and capabilities to be used as a general communication medium between processes *and* to obtain random variables quantities of major interest for parallel algorithms' complexity measures (e.g., *average* and worst-case values).

3) $A = \{ A_{p_1}, \cdots, A_{p_n} \}$ is the finite set of serial algorithms assigned to processes $p_1, \cdots, p_n$.

4) $S_I = \{S_{i_1}, \cdots, S_{i_k} \}$ is the finite set of initial states of processes.

5) $S_F = \{ S_{f_1}, \cdots, S_{f_n} \}$ is the finite set of final states of processes. The algorithm A terminates if every process enters a final state in S_F within *finite time* from the start of the algorithm execution (The algorithm can be considered starting whenever some process enters an initial state in S_I).

6) $E = \{ e_{p_1}, \cdots, e_{p_n} \}$ is a finite set whose i-th element e_{p_i} is the set of states of process p_i, $(1 \leq i \leq n)$.

7) $Q = \{ Q_1, \cdots, Q_r \}$ is a finite set such that $Q_j \subseteq (G_{\theta_j}, T_{\theta_j})$, where T_{θ_j} is the set of values taken by all processes variables within a unit time interval θ_j and $G_{\theta_j} = (\Pi_{\theta_j}, \Gamma_{\theta_j})$: $\Pi_{\theta_j} \subseteq P$ specifies the finite set of active emitting processes within a unit time interval θ_j, and Γ_{θ_j} specifies the finite set of connections linking pairs of processes in communication within a unit time interval θ_j. Thereby, the corresponding Q_j allows characterization of

a) messages *emitting* processes (set Π_{θ_j});

b) *immediate receiving* processes (of these messages). Hence, within a unit time interval θ_j the bidirectionnal connections are specified by the directed edges Γ_{θ_j} (for a given pair of processes, the edges' direction determines the emitting process and the message receiver);

c) the exact values taken by internal variables of all processes within a unit time interval (this is set T_{θ_j}).

8). $D \subset \Sigma$, (Σ given alphabet) D is then the finite set of accepted input data in the algorithm

A .

The above abstract model is a very complete one , especially with regard to "performance measures" of distributed algorithms. Hence, the whole model is actually not necessarily to be used entirely in each particular problem.

THREE FUNDAMENTAL PROBLEMS

With the same general approach, we make use of a weakened form of the above model (This weakened form we denote below a "model scheme") and we study three characteristic problems : Maximum finding, Merging and Sorting.

We do not refer to [A,K,S.] for example, though they present a sorting network of size $O(n\log n)$ and depth $O(\log n)$ (which is an excellent asymptotic result), because their algorithm requires a very large constant and seems quite complicated (as well as their model in other respects). Whereas, our basic instrument and reference are Valiant's algorithms [VAL.] which achieve (worst-case) optimal or nearly optimal bounds, though [VAL.] is counting only comparisons and does not take into account any over head at all.

Comparing our results to Valiant's [VAL.], Vishkin's [SH,VI.] and Kruskal's [KRU.] we show how they depend on the variations of our model scheme.

Shiloach and Vishkin [SH,VI.] implement Valiant's algorithm on a W-RAM and achieve Valiant's bounds but the W-RAM model is available for Valiant's algorithm of maximum-finding problem only (this is due both to conflicts of access to the central memory and allocation problems : that is, to allocate p processors to p tasks within a constant time). Kruskal [KRU.] implements Valiant's algorithms on a CREW P-RAM model and even improve on his (worst-case) results. Valiant's and Kruskal's improved algorithms are implemented on our model scheme that really takes into account *all* the communications issues : messages transmissions and overheads (Note that it involves neither allocation problems, nor fetch-and-add memory conflict between processes). Furthermore the implementation is much easier on our model than on the PRAM's.

PREMIERE PARTIE

CONCEPTS DE BASE, COMPLEXITES, EFFICACITE

We show that if the communication network is a clique (complete graph) Valiant's and Kruskal's optimal bounds are achieved in the worst case. Whereas, if the pattern of the communication network is not a clique - which is more realistic with respect to future machines - we show the evolution of bounds regarding the communication network capabilities.

These conclusions are somewhat different from Santoro's most recent results about distributed sorting [RO,SA,SI.]. This is because the latter model is a sheer distributed one, for sorting communication activities only; whereas ours is designed as stated above : a model scheme for parallel or distributed complexity measures in a specific sorting environment.

More precisely, in the worst case a clique given as communication network is shown to require $\Omega(1)$ messages, whereas a "grapes cluster" structure network is shown to require $\Omega(loglogp)$ messages (p= number of processes in the structure). So that, for the "n-elements maximum-finding" algorithm with p processors ([VAL.], [SV.] or [KRU.] make no distinction between processors and processes) we achieve Valiant's worst-case optimal bounds for $p \leq n$. and similarily for the merging of two sorted lists where our model scheme achieves Kruskal's worst-case bounds for $p < \lfloor n^{1-\frac{1}{k}} . m^{\frac{1}{k}} \rfloor$ ($k \geq 2$) and $2 \leq n \leq m$). At last, the model scheme achieves Kruskal's optimal bounds for the n-elements enumeration-comparison sorting (in the worst case) and, in particular, we show that an n-elements list can be sorted with $n.(\ logn\ )^{\frac{1}{k}}$ processes ($k \geq 2$) within $O(k.logn)$ time with our model scheme. This is an optimal upper bound worst-case result for enumeration-comparison sorting . (As a matter of fact, [SH,VI.]obtains an optimal upper bound of $O(k.logn)$ time, if $p = \lceil n^{1+\frac{1}{k}} \rceil$ ($k > 1$), in their comparison-sort parallel algorithm).

In a future paper, approximation algorithms of NP-complete problems as "Branch an Bound" for example will be considered with the help of the general abstract model A .

OOO

1. INTRODUCTION

L'apparition des machines paralleles multiprocesseurs asynchrones a haut degre de parallelisme et des grands reseaux distribues conduit a' repenser la conception des algorithmes.

Le terme "parallelisme" recouvre une realite diverse. Par ce mot, on entend aussi bien machines paralleles (quelles soient de type SIMD, ou MIMD, synchrones ou asynchrones, a flot de donnees, etc...) que reseaux distribues. Et ceci bien que les approches algorithmiques liees aux machines paralleles et aux systemes distribues soient qualitativement differentes, en particulier du point de vue des structures de donnees (et donc des acces y afferant): dans un cas, des donnees pouvant etre globales (cas des machines paralleles classiques), dans l'autre cas, les donnees etant, par nature uniquement locales (cas "distribue").

En ce qui concerne l'etude de la complexite des algorithmes induits par l'apparition du parallelisme en informatique, il nous a semble interessant de construire un modele permettant d'estimer l'efficacite (terme que nous preciserons par la suite) des algorithmes (et de preciser, en la fondant, la notion d'algorithme distribue) concus pour ces machines ou pour des reseaux.

La "complexite" des algorithmes paralleles ou distribues, est en fait une *mesure* relative a la fois au temps d'execution sequentiel, et au delai relatif au nombre de messages echanges durant l'execution generale de l'algorithme. Cette "mesure de complexite du parallelisme", nous la definirons comme *l'efficacite d'un algorithme parallele ou distribue.*

Ne nous interessant qu'a l'aspect de l'evaluation de l'efficacite des algorithmes, nous verrons comment on peut utiliser un modele unique pour estimer l'efficacite des algorithmes paralleles, et celle des algorithmes distribues, en considerant, en fait, de ce point de vue (et de ce point de vue seulement) les algorithmes paralleles comme un cas particulier d'algorithmes distribues.

Dans le cas sequentiel, la machine de Turing et la theorie des automates fournissent un modele formel pour:

a) donner un fondement theorique au concept d'algorithme;

b) en evaluer les performances en en mesurant la complexite.

Ceci entraine que dans le cas sequentiel, on peut concevoir les algorithmes de facon -relativement- independante des machines sur lesquelles ils seront executes. De meme, ce modele permet de batir une theorie de la complexite des algorithmes et des problemes dans laquelle la dite complexite ne depend que du modele formel, et pas du tout des machines reelles sur lesquelles sont executes les programmes codant les dits algorithmes.

Pour pouvoir etudier la "complexite" des d'algorithmes paralleles ou distribues, il faut proposer un modele formel a la fois detache des contraintes propres aux machines ou a tel ou tel type de reseau distribue et en meme temps pratiquement utilisable sur des machines reelles (actuelles *et* a venir).

C'est dans cet esprit que nous accordons de l'importance aux communications internes entre processeurs (et plus particulierement aux delais de communication) dans les *machines* paralleles, delais souvent negliges,

De meme, on doit tenir compte des "temps" de calcul des processus (ou pluto^t des processeurs qui hebergent les processus) dans un reseau distribue, en effet, ces temps de calculs ne sont pas necessairement negligeables devant les temps de transmission .

Par ailleurs, un modele d'algorithme parallele doit pouvoir se particulariser au cas sequentiel et fournir les memes resultats que la theorie classique concernant les algorithmes sequentiels.

2. PRECISIONS DE VOCABULAIRE.

2.1. Le concept de processus.

Le concept de processus est un concept abstrait par opposition a celui de processeur qui est une realite physique.

Dans ce qui suit, nous considererons donc qu'un processus est capable d'executer tout ou partie d'un algorithme sequentiel (deterministe ou non) selon ses communications avec les autres processus. Un algorithme distribue est alors vu comme une collection de processus communiquant entre eux, independemment du langage de programmation utilise chaque processus peut etre caracterise par *un ensemble d'etats* et par un ensemble de *transitions locales* liees aux mes-

sages echanges, de la forme [CAR.]:

$$(etat_k\ ,\ messagerecu) \dashrightarrow (etat_{k+1}\ ,\ messageenvoye)$$

Un processus peut executer un certain nombre d'instructions sequentielles (c'est a dire un certain nombre de transitions avec messages vides), et activer un ou plusieurs autres processus afin, soit de prendre a son compte la totalite d'un algorithme sequentiel (deterministe ou non), tout en activant le demarrage d'autres processus (tout ou partie des processus), ou simplement de leur communiquer une information (son etat, etc ...), soit d'executer un morceau d'algorithme sequentiel (eventuellement une attente) en commun avec d'autres processus a travers un reseau d'echange d'informations.

Le concept de processus repose sur l'execution d'un ensemble d'instructions sequentielles liees a l'echange organise d'informations avec d'autres processus.

Dans ce contexte, l'unite d'echange d'information ou de dialogue entre deux processus est le "message" et l'unite d'execution algorithmique est l' "execution d'une instruction sequentielle" (Un processus peut eventuellement ne rien faire pendant toute une partie de l'execution d'un algorithme parallele ou distribue)

. L'instruction sequentielle peut etre l'emission ou la lecture d'un message, l'activation d'autres processus, etc ...

2.2. Les communications entre processus.

Les communications entre processus se font par messages transitant a travers un reseau de communication *predetermine,* donne a l'avance, et qui est une caracteristique du systeme informatique utilise. Les regles suivant lesquelles sont achemines et delivres les messages constituent "le systeme postal" du reseau.

Nous supposerons ici que:

1) Tout message envoye par un processus p_i a un processus p_j avec lequel il est en communication (i.e. il existe une arete (p_i, p_j) dans le graphe representatif du reseau de communication) *arrive en un temps fini* caracteristique du systeme postal.

2) Le systeme postal n'altere ni ne modifie le contenu des messages (si on s'affranchit de cette propriete, on est confronte au probleme des generaux byzantins).

3) Si deux messages a et b sont envoyes dans un certain ordre, ab par exemple sur une meme ligne, a destination d'un processus p_j , ils arrivent dans le meme ordre, c'est a dire ab. (on peut ici aussi s'affranchir de cette contrainte en estampillant les messages [CAR.] mais cela compliquerait ici le propos sans rien apporter a l'etude.)

4) Chaque processus est muni d'une "boite aux lettres" ou file d'attente regie par une discipline de service FIFO, a un seul serveur (le processus lui meme) dont nous supposerons par exemple que la distribution des temps de service suit une loi exponentielle, et les arrivees de messages une loi de Poisson (i.e. les temps d'inter-arrivees sont des variables aleatoires exponentielles et independantes). Ceci afin de permettre l'utilisation des resultats deja connus sur les files d'attente (sur ce sujet, voir [GE,PU.]).

Selon la notation usuelle de Kendall, on peut caracteriser une telle file comme M/M/1/C/FIFO.

Remarque 1 : Dans la suite, le reseau de files d'attente est *discret* . Mais (voir Kobayashi), un processus de Poisson possede deux analogues discrets : la *suite de Poisson* et la *suite de Bernoulli* (chacune de parametre λ, par exemple). La premiere est telle que $Pr\{ X_k = n \} = \frac{\lambda^n}{n!} . e^{-}\lambda,\ k = 0,1,2,...$ (X_k variables aleatoires discretes independantes de la suite). La seconde est telle que $Pr\{X_k = 1\} = \lambda; Pr\{X_k = 0\} = 1 - \lambda,\ k = 0,1,2,...$ (X_k v.a. discretes independantes a valeurs dans $\{0,1\}$).

Remarque 2 : On peut considerer aussi une file M_x/M/1/c/FIFO ou M_x signifie que les arrivees de messages peuvent se faire en groupe; x etant alors la variable aleatoire representant le nombre de messages par groupe dans une telle arrivee. Chaque file devant etre de longueur bornee a tout instant, c represente la capacite *maximale* de la file.

On peut prendre en compte l'arrivee de messages prioritaires (messages en transit par exemple). La priorite de tels messages sera consideree comme simple, c'est a dire qu'un message prioritaire (simple) n'est traite avant un autre, non prioritaire que dans la mesure ou le

serveur-processus a termine le traitement-service du message precedent (pas de preemption).

Il est naturel de definir la priorite d'un message en fonction directe du nombre de services qu'il a deja subit[2].

Le message le plus prioritaire de la file (i.e. celui dont le niveau de priorite est le plus eleve), est alors celui qui a effectue le plus long trajet dans le reseau.

On est donc amene a considerer, pour chaque processus, une file d'attente constituee de classes distinctes de clients, suivant leurs niveaux de priorite (file d'attente a plusieurs classes de priorite).

Remarque 3 : Suivant la nature de l'algorithme lui meme, il peut y avoir des classes distinctes de messages en priorite absolue, c'est a dire que de tels messages sont traite "toute affaire cessante" si leur niveau de priorite est superieur a celui du message en cours de traitement. Le serveur peut, si son etat le permet, reprendre le traitement du message qui etait en cours de traitement au point ou il l'avait laisse a l'arrivee du prioritaire lorsque le traitement dudit prioritaire est termine.

Remarque 4 : Une autre facon de traiter le probleme serait de considerer que chaque processus possede une boite a lettres (i.e. une file d'attente) pour chaque autre processus avec lequel il est relie sans intermediaire (i.e. il existe une arete dans le graphe representatif du reseau, entre les sommets figurant des processus pouvant communiquer sans intermediaire). Dans la meme optique, chaque file d'attente pourrait etre consideree comme etant "a double entree" selon la technique developpee par N. Santoro [AT,SA,SA,ST.].

2.3. Le reseau markovien de files d'attente.

Le modele decrit ci-dessus definit un reseau de communication $G = (P , L)$ dont les sommets sont les n processus (ensemble P) *et les arcs (eventuellement, on peut considerer qu'entre deux processus il y a deux arcs orientes de sens contraire. Dans ce cas, par abus de langage, on parlera d'arete pour le graphe associe), ou lignes de communication entre ces processus, les ele-*

[2] C'est exactement le procede de calcul des priorites qui est utilise dans le reseau d'interconnexion de la machine HEP 2 de Denelcor, chaque fois qu'un message est deroute du chemin optimal devant le conduire a destination, son niveau de priorite augmente de une unite.

ments de L. G est un graphe sans circuit, et sans arcs multiples. Chacun des sommets-processus etant muni d'une file d'attente de type M/M/1/c , (c etant la capacite maximale d'une file). G definit ainsi un reseau de files d'attente (celles-ci pouvant etre considerees comme *discretes* . cf. Kobayashi [LO,LA.]) avec les proprietes suivantes:

P1- La discipline du reseau G est de type M/M/n/n.c (n.c est alors la capacite maximale du reseau). La discipline de service peut etre FIFO ou bien, comme evoque precedemment, toute discipline avec files prioritaires. Cet aspect n'affecte en rien les potentialites du modele.

P2- Le reseau de files d'attente G est ferme, dans le sens suivant: entre deux instants consecutifs donnes, ou bien, durant un intervalle de temps unitaire θ donne, le nombre maximal total de messages emis est n . Par consequent, le systeme postal de routage dans G, l'attente et le service sont bornes dans G.

P3- Les n serveurs processus du reseau G ont une distribution des temps de service exponentielle de taux μ_i ($1 \leq i \leq n$) et une distribution des arrivees de messages poissonienne de parametre λ_i ($1 \leq i \leq n$) (voir Kobayashi dans [LO,LA.], pour l'equivalent *discret* de ce modele et les demonstrations y afferant). λ_i *et* μ_i sont des parametres qui dependent du nombre de clients-messages en attente en chaque serveur-processus (eventuellement, ils sont aussi fonction de la taille des dits messages; mais on peut toujours s'affranchir de cette hypothese en majorant la taille de tout message envoye sur un intervalle de temps θ).

P4- De P1, P2, et P3, on deduit que G est un reseau ferme markovien M/M/n/n.c. C'est donc un modele classique de processus (au sens de Markov) de renouvellement avec etat d'equilibre et solutions a "forme produit". En l'occurence, le reseau est de "type BCMP" (voir [GE,PU.] et [LO,LA.] pour la definition d'un reseau BCMP et le theoreme y afferant).

A partir des proprietes ci-dessus enoncees, on montre facilement que G est un reseau de files d'attente *markovien* (voir [FEL.]).

Preuve: Si $t_{n-1} < t_n$, alors, X(t) etant une variable aleatoire, on a: $P\{X(t_n) \leq x_n \mid X(t), t \leq t_{n-1}\}$. Il s'ensuit que, si $t_1 < t_2 < \cdots < t_n$, on a $P\{X(t_n) \leq x_n \mid X(t_{n-1}), \cdots, X(t_1)\} = P\{X(t_n) \leq x_n \mid X(t_{n-1})\}$. ceci est encore vrai pour un processus en temps *discret* si on remplace $X(t_n)$ par X_n .

Ainsi, comme nous le montrerons plus loin, notre modele est asses puissant pour etre utilise de deux facons, ou considere comme un reseau a caractere double:

D'une part comme un reseau d'intercommunication entre les processus ("reseau postal" par BUS par exemple)

D'autre part comme un reseau ferme markovien de files d'attentes, ce qui permet d'obtenir des resultats sur les variables aleatoires qui sont tres utiles pour la *"mesure de complexite des delais de communication"*: par exemple la moyenne des temps de service, la moyenne des temps passes par les messages dans le reseau, etc...

Par ailleurs, ce modele suggere d'autres voies de recherche comme:

- Dans le domaine probabiliste, avec eventuellement des arguments de type stochastique sur le modele, ou bien meme des possibilites d'etude d'algorithme distribues probabilistes afin de determiner des mesures de complexite en moyenne (voir [VIS.2], [RE,VA.], [UPF.] ou [YAO]); et une utilisation de la theorie des files d'attente et du renouvellement (voir les travaux recents de Kruskal [KR,WE.]) pour obtenir, par exemple, des solutions lors de la recherche d'une strategie optimale pour le probleme d'execution de taches composees de plusieurs sous-taches independantes: il faut alors synchroniser les processus et, en un minimum de temps, traiter la tache en question lorsque toutes les sous-taches en ont ete traitees (voir [KR, WE.]).

2.4. Distribue et Parallele.

L' *execution* d'un algorithme parallele peut etre consideree comme le travail d'un ensemble de processus executant chacun un algorithme sequentiel qui lui est propre (eventuellement le meme) et echangeant des messages entre eux a travers un reseau de communication donne. La nature du reseau des communications entre les processus, c'est a dire le graphe des communications "directes" (i.e. sans passer par un processus intermediaire) des processus deux a deux, est induite par la facon dont s'effectuent ces communications; par donnees partageables (donc par memoire commune) ou par messages.

Afin de se ramener a un modele unique pour l'etude de l'efficacite des algorithmes paralleles ou distribues, on peut, dans le cas d'une memoire partageable considerer, soit que cette memoire est un processus particulier, ce qui conduit a un graphe "en etoile" [3] soit faire abstraction de cette memoire, et considerer que tout processus p_i peut communiquer avec tout processus p_j, ce qui nous conduit a considerer que dans ce cas, tout se passe comme si les processus communiquaient a travers un reseau de communication pour lequel le graphe associe serait une clique.

Par consequent, nous allons construire un modele unique dans lequel la distinction entre "parallele" et "distribue" tiendra a la nature de ce graphe representatif des possibilites de communications des processus deux a' deux.

Ce modele cherchant a prendre en compte les phenomenes influant sur l'efficacite des algorithmes lies tant aux machines paralleles qu'aux reseaux ne pourra etre utilise que rarement dans sa totalite pour l'etude de l'efficacite d'un algorithme particulier. Ainsi, dans la deuxieme partie, nous examinerons comment evoluent les bornes de quelques algorithmes de recherche du maximum, de tri et de fusion qui ont ete concus pour des machines paralleles, et pour ce faire, nous n'utiliserons qu'une partie du modele, la prise en compte de la topologie du graphe des communications (ce qui est totalement ignore par les modeles du type PRAM, mais qui est

[3] Ce modele correspond au modele dit PRAM ([SH,VI.], [VIS.1]), dans le quel une batterie de RAM se partagent une memoire commune. On distingue plusie urs types de PRAM (WRAM, CREW-PRAM, etc... : [BO,HO.], [BLU.]) en fonction des proprietes supposees des operations de lecture et d'ecriture sur ces modeles. Cette distinction met bien en lumiere les problemes d'implementation qu'elles posent tels les possibles conflits d'acces a la memoire commune, la reallocation de processeurs a des taches ou des sous-taches, etc... .

susceptible d'influer sur les resultats d'efficacite, comme nous le verrons). Dans cette etude, tout l'aspect lie a la gestion des files d'attentes des messages echanges est laisse de cote. Par contre, dans l'etude de l'efficacite d'un algorithme d'enumeration implicite (voir [LA,RO.1]), nous verrons comment la gestion des files d'attente doit etre prise en compte, et ce que cela signifie.

Comme le fait remarquer Vishkin [VIS.1]:

"A traditionnal computer science technique used to support the claim of robutness of an abstract model of computation (or a complexity measure of a resource within some model of computation) is to show that, for reasonably defined model they can simulate one another with a complexity measure preserved by the simulation".

Nous placant de ce point de vue, nous montrerons la pertinence de ce modele pour l'evaluation de l'efficacite d'algorithmes paralleles connus, sur la base de ceux de Valiant [VAL.], pour lesquels Kruskal, Shiloach et Vishkin [SH,VI.] etc ... ont construit des modeles ad-hoc, et nous montrerons ce qu'il advient des bornes de complexite qu'ils ont calculees lorsqu'on prend en compte des contraintes liees a la realite des systemes informatiques, comme dans notre modele.

Le modele de la PRAM de ce point de vue, est, a notre avis trop eloigne des realites d'implementation pour etre un bon outil de l'evaluation de l' "efficacite" des algorithmes. On peut encore utiliser le modele de la PRAM pour les machines existantes, mais pour les machines paralleles a venir, a' parallelisme massif, ce modele n'est plus pertinent, il est en effet exclu que des machines comportant un grand nombre de processeurs (en l'etat actuel de la technique, les machines paralleles existantes ne peuvent supporter plus de 16 processeurs) n'utilisent pas un reseau d'interconnexion, donc de communication. De plus, la necessite d'un acces rapide aux donnees conduit de plus en plus dans de telles architectures a une decentralisation des donnees dans les processeurs.

En fait, c'est la que se situe le caractere le plus discriminant entre algorithme parallele et algorithme distribue, du moins pour l'etude de leur efficacite.

On est dans le cas "distribue" lorsque les donnees sont locales, lorsqu'il n'y a pas de variable globale. C'est cette propriete qui fait que dans le cas distribue les communications entre processus se font exclusivement par message, alors que dans le cas d'une machine parallele, les communications se font par l'intermediaire de variables partageables, ou globales. L'evolution previsible des machines parallele permet de penser que, le nombre de processeurs se multipliant, elles auront des architectures totalement ou partiellement distribuees (voir par exemple Cyber XX de CDC). C'est bien alors le mode de communication qui distingue les algorithmes destines aux machines paralleles actuelles des algorithmes distribues.

C'est pourquoi, lorsqu'on parle d'algorithme parallele, on entend en general, algorithme pour machine parallele. Par contre lorsque l'on evoque le parallelisme d'un algorithme, c'est pour mettre en evidence que certaines parties sont suffisement independantes pour quelles soient traitees simultanement ou non (de facon synchrone ou non), c'est a dire que leurs successions temporelles soient relativement independantes.

2.5. Intervalle de temps observable.

Appelons θ un intervalle de temps pendant lequel un processus quelconque peut *changer d'etat* c'est a dire, soit envoyer un message et un seul, soit executer une instruction sequentielle (eventuellement d'attente). Il est evident que durant un tel intervalle de temps observable, *plusieurs* processus peuvent changer d'etat, par exemple envoyer un message (mais un seul par processus). Un tel intervalle de temps sur lequel au moins un processus change d'etat et aucun ne change deux fois d'etat, sera appele intervalle de temps observable, ou encore intervalle de temps unitaire.

Par la suite, nous noterons Θ l'ensemble $\{\theta_1, \theta_2, \cdots, \theta_t\}$ des differents intervalles de temps observables durant l'execution de l'algorithme.

Si on note d_i la *duree* de l'intervalle θ_i , la duree totale d'execution d'un algorithme parallele est alors:

$$T = \sum_{i=1}^{i=t} d_i .$$

Et en particulier, si on considere les θ comme des unites de temps (intervalles unitaires), on a alors:

$$T = |\Theta| ; (|\Theta| \text{ signifiant Cardinal de } \Theta).$$

On peut maintenant aborder la definition formelle d'un algorithme distribue.

3. DEFINITION D'UN ALGORITHME DISTRIBUE.

3.1. Definition:

Nous appelons *Algorithme distribue* l'octuplet suivant [4]

$$A = (P, G, A, S_I, S_F, E, Q, D)$$

dont les differents constituants sont:

1) $P = \{p_1, p_2, \cdots, p_n\}$ l'ensemble des processus necessaires.

2) $G = (P, L)$ le graphe (non oriente, sans cycle ni aretes multiples) des communications possibles entre les processus; les sommets de G sont donc les processus, et les aretes de G representent les *possibilites* de communication directe (c'est a dire sans intermediaire) de processus a processus. On notera E l'ensemble des etats du reseau markovien ferme M/M/n/n.c attache a G

3) $A = \{A_{p_1}, A_{p_2}, \cdots, A_{p_n}\}$ l'ensemble fini des algorithmes sequentiels attaches aux processus $p_1, p_2, \cdots, p_n$.

4) $S_I = \{s_{i_1}, s_{i_2}, \cdots, s_{i_n}\}$ l'ensemble fini des etats initiaux des processus.

5) $S_F = \{s_{f_1}, s_{f_2}, \cdots, s_{f_n}\}$ l'ensemble fini des etats finaux des processus (voir en Remarque les conditions de depart et d'arret d'execution de l'algorithme A).

6) $E = \{e_{p_1}, \cdots, e_{p_n}\}$ l'ensemble fini dont le i-eme element, e_{p_i}, est l'ensemble des etats pris par le processus p_i sur tous les intervalles θ, $(1 \leq i \leq n)$

Considerons par exemple le graphe de la figure 1

---

[4] En fait, s'il s'agit bien la d'un modele formel d'algorithme distribue ce modele n'est pas utilisable pour definir les algorithmes destines aux machines paralleles actuelles, car il ne rend pas compte des structures de donnees globales possibles sur ces machines. Toutefois, pour l'etude de l'efficacite des-dits algorithmes, ce modele convient tout a fait car une telle etude ne prend pas en compte la *structure* des donnees (voir [SAN.] et [RO,SA,SI.], pour une approche tres semblable du probleme - mais centree uniquement sur des mesures de complexite en messages).

fig 1

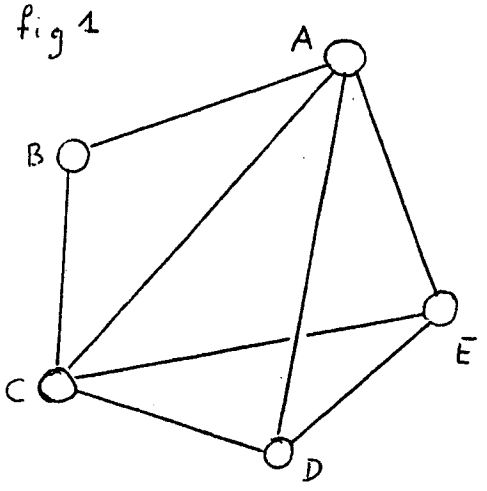
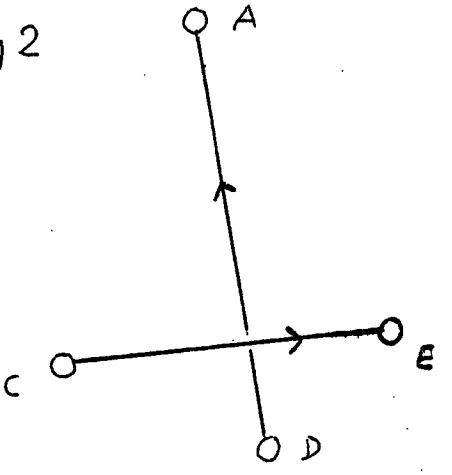


fig 2



$P = \{A, B, C, D, E\}$, et, $G. = (P, L)$.

7) $Q = \{Q_1, \cdots, Q_r\}$ un ensemble fini tel que $Q_j \subseteq (G_{\theta_j}, T_{\theta_j})$ *ou* T_{θ_j} est l'ensemble des valeurs prises par les variables de tous les processus sur l'intervalle θ_j, $j \in J = |\Theta|$ et $G_{\theta_j} = (\Pi_{\theta_j}, \Gamma_{\theta_j})$ *ou* $\Pi_{\theta_j} \subseteq P$ (c'est l'ensemble des processus emettant sur l'intervalle θ_j).

Par exemple, sur l'intervalle θ_j , on peut avoir la situation decrite par la figure 2, et on a; $\Pi_{\theta_j} = (A, C, D, E)$, et , $\Gamma_{\theta_j} = ((C,E) ; (D,A))$, ce qui signifie que les processus emetteurs sont C et D .

En fait, sur un intervalle de temps θ_j, le Q_j asocie permet de connaitre:

a) les processus emetteurs de messages (c'est le role devolu a Π_{θ_j});

b) les processus immediatement destinataires de ces messages (immediatement signifiant ici que deux processus ne peuvent communiquer entre eux que si les sommets leur correspondant dans G sont relies par une arete). Ces liaisons sur un intervalle θ_j sont alors representees par les *arcs* Γ_{θ_j} (l'orientation de l'arc est a noter car c'est elle qui permet de connaitre le processus emetteur et le processus recepteur lors d'un echange).

c) les valeurs prises par les variables internes de tous les processus pendant un intervalle observable θ_j , cet ensemble etant T_{θ_j} .

Ainsi, sur un intervalle θ_j , la connaissance de Q_j permet de caracteriser l'etat d'avancement de l'execution de l'algorithme distribue etudie.

8) $D \subset \Sigma$. (Σ alphabet donne); D est alors l'ensemble des donnees d'entree acceptees par A pour effectuer un calcul.

3.1.1. Nature du graphe des communications effectuees sur un intervalle observable.

Le graphe $G_{\theta_j} = (\Pi_{\theta_j}, \Gamma_{\theta_j})$ est un graphe oriente par l'emission de messages entre processus sur un intervalle de temps observable θ_j , or, par definition de la notion d'intervalle de temps observable, durant θ_j un processus ne peut emettre qu'un seul message.

Par consequent:

$$(\forall \Pi \in \Pi_{\theta_j}) \quad d^+\Pi \leq 1$$

ou d^+ designe le demi-degre exterieur d'un sommet;

On peut donc enoncer le theoreme:

Theoreme :

$(\forall \theta \subset \Theta)$ $G_{\theta_j} = (\Pi_{\theta_j}, \Gamma_{\theta_j})$ est un graphe fonctionnel.[5]

3.2. Cas sequentiel.

Le cas sequentiel classique est celui ou $P = 1$, l'octuplet se reduit alors au quadruplet:

$$A=(S_I, S_F, E, D)$$

ou:

- A est l'algorithme,
- S_I et S_F respectivement les etats initial et terminal.
- E l'ensemble des etats intermediaires,
- D les donnees.

On retrouve la la notion sequentielle classique.

COMPLEMENTS :

a) Soit $d \in D \subseteq \Sigma$; d est accepte avec une complexite en temps $\in_t$ si, et seulement si, A, place

[5] Toutes les notations sur les graphes sont, sauf mention explicite, celles de [BER.]

dans un etat de S_I s'arrete au bout d'un temps $\in_t$ fini dans un etat de S_F.

b) L'execution de l'algorithme A demarre des qu'un processus quelconque entre dans un etat initial de S_I. L'execution de A se termine, lorsque tout processus de P entre dans un etat final de S_F *au bout d'un temps fini* (depuis le debut de l'execution de A); le calcul est alors termine.

c) Il est toujours possible de rendre un algorithme parallele A non deterministe, ou meme "alternant", soit en considerant les $A_i \in A$ comme des algorithmes sequentiels non deterministes dans le premier cas, soit en considerant dans A une application (voir [KOZ], [PA,PR,RE.],[PA,RE.],[CH,KO,ST.]).

$$f : Q \longrightarrow \{ \wedge , \vee \}$$

avec $f(Q_j) = \text{'et'} \longrightarrow$ etat universel
$f(Q_j) = \text{'ou'} \longrightarrow$ etat existentiel.

Les definitions se rejoignent en notant: ND A = Alt A tel que: $f(Q_j)=$'ou' (uniquement!) et en omettant f dans A. On peut alors utiliser la puissance du non-determinisme dans le parallelisme.

d) On fait implicitement une hypothese generale de repartition probabiliste des algorithmes A_i sur l'ensemble des processus (par exemple, de repartition uniforme : loi uniforme sur [1,n]), ainsi que des messages entre processus.

Pour rester coherent avec la distribution des arrivees de messages dans les files d'attente des processus, laquelle est supposee poissonienne, on peut faire l'hypothese que la repartition des messages echanges entre les processus de P est une repartition suivant une *loi de Poisson sur les aretes* du graphe G du reseau de communication. Ici, on considerera en fait que chaque arete du graphe est composee de deux arcs orientes de sens contraire.

En prenant alors en compte cette double orientation possible: (p_i , p_j) et (p_j , p_i) $(\forall\ i,j\ \ 1 \leq i,j \leq n)$. , le nombre d'aretes $|L|$ est tel que : $2.(n-1) \leq |L| \leq 2.\binom{n}{2}$

Des hypotheses de repartition differentes pour les A_i (non uniforme ou uniforme sur les n! permutations de P) et pour les messages (repartition non poissonienne, mais uniforme sur les $|L|$ permutations des aretes avec leur double orientation possible) donneraient des resultats

fort differents par la suite ("en moyenne" en particulier).

Quoiqu'il en soit, quelles que soient les hypotheses de repartition probabiliste choisies, l'essence du modele *A* n'en est pas affectee.

3.3. Algorithme parallele.

Bien que, comme nous l'avons vu precedemment, il n'y ait pas de difference fondamentale entre un algorithme pour machine parallele et un algorithme distribue, du moins du point de vue ou nous nous plaçons, nous faisons tout de meme une mention speciale de ceux-ci, car les machines paralleles existant actuellement nous y conduisent.

Deux cas peuvent se presenter en algorithmique parallele.

A) On veut utiliser un ensemble de processeurs sans memoire commune partageable relies a travers un reseau de communication bien determine, permettant aux dits processeurs de dialoguer entre eux, les communications *ne pouvant* se faire de tout processeur a tout autre sur un seul intervalle de temps observable. Peu importe qu'il s'agisse alors d'un reseau de processeurs au sein d'une meme machine, ou d'un reseau de machines geographiquement eloignees les unes des autres, l'aspect discriminant etant qu'il n'y ait pas de possibilite de variable globale.
Il s'agit alors de construire et de specifier des algorithmes distribues, la topologie du reseau etant tres importante tant du point de vue des performances de l'algorithme que pour la facon de transmettre les messages (i.e. le protocole de communication). En ce cas, en un intervalle de temps θ, deux processus ne peuvent communiquer que s'il existe physiquement une ligne de communication entre les processeurs leur correspondant (attention toutefois au fait que si la topologie du reseau est determinante pour les performances de l'algorithme, l'ecriture, et la conception d'un algorithme distribue sont independantes de celle-ci.)

B) On suppose qu'il existe une memoire commune partageable par tous les processeurs, alors, tout processeur peut communiquer avec tout autre sur un intervalle de temps observable θ. Peu importe alors la facon dont se font les communications entre les processeurs (cross-bar,

memoire partageable, etc ...) on considere alors que le graphe G des communications est soit une clique, soit une "etoile" [6] (voir figure).

Nous considerons donc qu'un algorithme parallele est un algorithme distribue dans lequel le graphe G des communications est soit une clique, soit une "etoile".

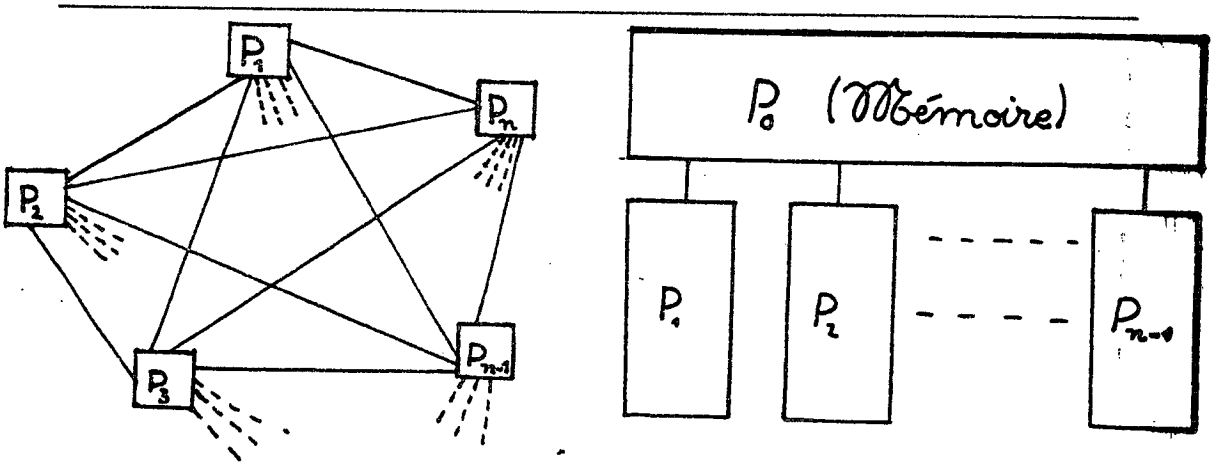
4. MESURES DE COMPLEXITES.

Dans le cas du parallelisme, pour evaluer les performances des algorithmes, il nous faut prendre en compte plusieurs parametres. Les parametres a notre avis les plus importants sont:

a) le nombre de messages echanges durant le temps d'execution de l'algorithme

b) le nombre d'unites de temps ou de "pas" de calcul necessaires a l'execution de l'algorithme.

En effet, on ne peut se contenter d'un seul de ces deux parametres, contrairement a une idee largement repandue. On peut avoir un algorithme dont la complexite en temps soit "bonne" (i.e. polynomiale) mais pour lequel le nombre de messages echanges soit prohibitif, et provoque un allongement demesure de la file d'attente de certains processus. A l'inverse, on peut concevoir des algorithmes pour lesquels le nombre de messages echanges est faible, mais dont la complexite en temps pour un processus est prohibitive (exponentielle par exemple),



[6] La structure en etoile nous ramene au modele de la RAM, et peut se ramener simplement a celui de la clique en raisonnant sur des intervalles de temps de 2 Θ.

c'est le cas entre autres des algorithmes distribues pour des procedures d'enumeration implicite, telles les PSES ou les PSEP (voir [LA,RO.2]).

Les travaux de Santoro en particulier, [SAN.] et [RO,SA,SI.] par exemple, explorent methodiquement une direction de recherche : la "complexite en nombre de messages" dans les algorithmes distribues. Cependant, ces resultats ne sont valides que dans le cas purement distribue, sur un modele tres epure de reseau de communication ([RO.SA.SI.] ne prend en compte que des sites munis d'une memoire locale non partageable). Ceci explique que les resultats de [RO.SA,SI.] et les notres (Partie II) different sur le rapport entre, d'une part, "le temps de calcul local" et, d'autre part, les delais de transmissions et d'attente dans les files dans le cas *concret* d'une analyse de tri distribue sur une liste de N elements (sur des sites pour [RO,SA,SI.], et a l'aide de processus pour nous).

A la difference du modele distribue "ideal" de [RO,SA,SI.] ou le cou^t des transmissions de messages est suppose dominer tous les autres, nous utilisons pour l'analyse des mesures de complexite, tous parametres pris en compte, un algorithme de tri *tres performant* en temps de calcul et un modele parallele ou distribue *particularise au cas d'une machine parallele* afin d'optimiser le cout en temps d'echange de messages dans le pire des cas, du point de vue du modele.

Les buts et les hypotheses etent different en theorie, les deux classes de resultats sur un exemple concret ne peuvent etre significativement comparees ni opposees; elles ont un interet et une justification plus ou moins profonds, mais seulement dans leur domaine de validite specifique.

4.1. Definitions et notations.

Les mesures de complexite τ, ξ, η de l'algorithme *A que nous introduisons ici sont des mesures discretes de complexite en temps.*

Nous faisons la distinction fondamentale suivante entre deux variables aleatoires entieres independantes :

1) *le nombre d'instructions executees* par chaque algorithme A_i $(1 \leq i \leq n)$. (Ce nombre comprend le "traitement" de messages recus qui n'est autre que l'execution d'instructions), v.a. entiere *uniformement distribuee* sur A (i.e. sur [1,n])

et

2) *le nombre de messages emis* par chaque processus p_i $(1 \leq i \leq n)$, v.a. entiere distribuee sur P selon une *suite de Bernouilli* de parametre adequat (dependant de n et des specificite du probleme). Ce nombre comprend les messages receptionnes en file d'attente suivant, par exemple, les disciplines de service definies plus haut.

Ainsi, pour toute donnee $d \in D$, on definit :

i) $\tau\ A_i(d)$ comme une variable aleatoire de cout en temps d'execution de toutes les instructions de chaque algorithme A_i , relativement a la mesure discrete τ.

ii) $\xi\ p_i(d)$ comme une variable aleatoire de cout en temps d'emission de tous les messages relativement a chaque processus p_i , relativement a la mesure discrete ξ.

iii) $\eta\ A(d)$ comme une variable aleatoire de cout en temps total des messages *et* instructions de l'algorithme parallele, relativement a la mesure discrete η. ($\tau A_i(d)$ *et* $\xi p_i(d)$ sont deux variables aleatoires entieres independantes.)

Remarques:

a) Ces mesures sont ici discretes afin d'utiliser un reseau de files d'attente discret - voir [LO,LA.] pour la correspondance demontree par Kobayashi entre discret et continu/loi de Bernouilli et loi de Poisson.

b) Ces mesures peuvent bien entendu etre eventuellement ponderees par une ou des constantes liees a une machine donnee, ou a un reseau particulier.

c) Toutes ces variables aleatoires de cout sont exprimees en temps ou en "pas" de calcul des algorithmes A_i , en temps des messages des processus p_i et en temps global de A respectivement, en considerant comme unite de temps commune un intervalle observable θ .

On introduit $\eta\ p_i(d)$ comme etant une variable aleatoire de cout en temps total en nombre de messages *et* instructions du processus p_i , relativement a^t la mesure discrete η.

4.2. Complexite en temps pour un processus dans le pire des cas.

Si on note t le temps durant lequel un processus est "vivant", c'est a dire le nombre de θ qui s'ecoulent entre le demarrage du processus et le moment ou il s'arrete (correctement) definitivement, on aura alors:

$$t(p_i) = \sum_{j \in K} \theta_j \; ; \; K \subseteq J \quad (J \text{ etant l'ensemble des intervalles } \theta_j).$$

et plus precisement, pour un processus p_i , dans le pire des cas, on aura:

$$\eta_{pire}(p_i(d)) = \tau A_i(d) + \xi(p_i(d)) \; (\forall \; d \in D_m)$$

c'est a dire la somme des couts d'execution de l'algorithme A_i dans le pire des cas, et du cout en nombre de messages emis par le processus p_i . C'est a dire encore que le parallelisme n'agit pas, tout se passe sequentiellement.

4.3. Complexite en temps, dans le pire des cas, d'un algorithme distribue (ou parallele).

Si on note comme ci-dessus $\eta(p_i(d))$ la complexite en temps du processus p_i pour une donnee d, et ce dans le pire des cas, on notera:

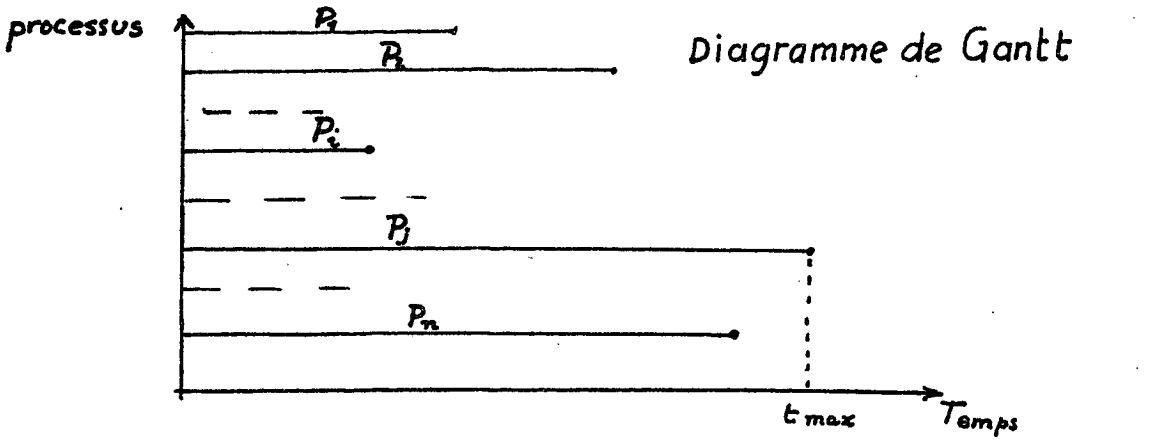
$$\eta_{pire}(A(m)) = \underset{i}{Sup}(\underset{d}{Max}\{\eta_{pire}(p_i(d)) \mid d \leq m \; ; \; d \in D\})$$

C'est le temps mis par le processus le plus lent sur la donnee la plus defavorable; en termes d'ordonnancement, *c'est la date au plus tard* de la terminaison .

La complexite en temps dans le pire des cas pour l'algorithme distribue (ou parallele) A.

Representons sur un diagramme du type diagramme de Gantt (figure 2), l'ordonnancement dans le temps du fonctionnement des processus p_i consideres comme des taches independantes d'un probleme d'ordonnancement (on ne fait pas figurer les communications sur le diagramme, mais les temps correspondants sont supposes pris en compte).
Sur ce diagramme, on constate que le temps total de calcul (au sens large) de l'algorithme A est egal au temps de calcul le plus long parmi ceux des processus de l'algorithme A pour une donnee de taille $d \leq m$.

Remarque :Dans tout ce qui precede, comme dans tout ce qui suit, nous supposons que tous les processus demarrent en meme temps.



On peut toujours se ramener a ce cas, il suffit de considerer qu'un algorithme A a demarre des lors que l'un au moins de ses processus est actif, tous les autres processus etant alors consideres comme ayant demarre, meme s'ils sont, pendant un certain temps en attente.

4.4. Complexite moyenne en temps $E[\eta A(m)]$

A partir des variables aleatoires independantes $\tau\ r{:}A(m)$ et $\xi\ r{:}P(m)$ (respectivement, le cout total d'execution de tous les algorithmes A_i et le cout total d'emission de messages pour tous les processus p_i - et cela pour toute donnee de taille m), on peut prolonger l'etude dans deux directions au moins:

1) Calculer $E[\tau\ A(m)]$: a l'aide des series generatrices de cou^t, l'esperance (et la variance) des $\tau\ r{:}A_i(m)$ puis de $\tau\ A(m)$ peuvent se deduire d'une analyse combinatoire initiale de l'algorithme.

2) Evaluer $E[\xi\ r{:}P(m)]$ comme temps moyen de service dans le reseau de files d'attente G, reseau markovien type M/M/n/c par exemple (cf. [GE, PU.]) grace a la formule de Little: $E[N] = \lambda\ .\ E[\xi r{:}P(m)]$ (ou $E[N]$ = nombre moyen de messages dans le reseau, et λ = parametre de la loi de Poisson regissant les arrivees de messages dans les files de priorite du reseau).

On peut calculer $E[N]$ par exemple a parir des equations de regime stationnaire, les equations de Chapman-Kolmogorov,

puis,

3)conclure par la complexite en moyenne $E[\eta A(m)]$ avec

$$E[\eta A(m)] = E[\tau A(m)] + E[\xi r{:}P(m)] \quad (\forall \quad m = |d|, \ d, \text{tailleduprobleme}).$$

4.5. Complexite en temps dans le meilleur des cas.

Reprenons le diagramme de Gantt de la figure 2, si on suppose que chaque processus p_i execute, sur une donnee d, l'algorithme A_i qui lui est associe, et ce dans le meilleur des cas; alors le temps total pris par l'execution de A est le plus grand des temps d'execution des A_i . D'ou la definition:

Definition : On appellera complexite en temps dans le meilleur des cas pour un algorithme A s'executant sur r processus, qu'on notera $\nu A(\mathrm{d})$; d etant une donnee de taille inferieure ou egale a m :

$$\nu A(\mathrm{d}) = \underset{i=(1,\ldots,r)}{Max} (Min\{\eta p_i(\mathrm{d}))| \ |\mathrm{d}| \leq \mathrm{m}\})$$

4.6. Efficacite de l'algorithme *A pour un probleme.*

Des donnees precedentes: complexite dans le pire des cas, complexite en moyenne, puis complexite dans le meilleur des cas, on peut definir ce que nous nommerons "l'efficacite de A", *notee* $\Psi(A)$. :

$$\Psi(A) = < \eta_{pire}(A); \ \mathrm{E}[\eta(A)]; \ Var[\eta(A)]; \ \nu(A); \ \alpha(A) >$$

ou $Var[\eta(A)]$ est la *variance* de $\eta A(\mathrm{d})$ (ou moment d'ordre deux de la variable aleatoire $\eta A(\mathrm{d})$ connaissant son moment d'ordre un: $\mathrm{E}[\eta(A)]$) et $\alpha(A)$ le "speed-up" de A *pour un probl/*"eme.

$$\alpha(A) = \frac{\mathrm{E}[\tau(\mathrm{A})]}{\mathrm{E}[\eta(A)]}$$

Ou $E[\tau(\mathrm{A})]$ *represente la complexite sequentielle en moyenne correspondante (i.e. celle obtenue avec un seul processus).*

Remarque : On peut obtenir des resultats asymptotiques significatifs sur $\Psi(A)$ en faisant tendre

la taille de la donnee vers l'infini. On peut en deduire des bornes superieures et inferieures de la complexite de A, en particulier en moyenne, ainsi que pour l'acceleration (ou "speed-up").

DEUXIEME PARTIE

APPLICATION A QUELQUES PROBLEMES CLASSIQUES

○

RECHERCHE DU MAXIMUM, FUSION TRI

1. INTRODUCTION

Le modele decrit dans la premiere partie est tres general, et sa particularisation a l'etude de tel algorithme, ou tel autre, fait que ce modele est rarement utilisable dans sa totalite. Ainsi, pour les algorithmes etudies dans cette partie, il n'est nul besoin d'utiliser les files d'attentes, ni l'ensemble Q. Par contre, pour pouvoir etudier des algorithmes distribues d'enumeration implicite (PSES et PSEP), comme dans [LA,RO.] cette partie du modele est fort utile, de meme que l'etude du comportement des elements de l'ensemble Q peut etre necessaire pour etudier le fonctionnement d'un algorithme distribue tel celui d'arbre couvrant minimal dans un graphe, comme en [LA,RO.1]. Nous allons montrer ici comment on peut etudier avec ce modele l'efficacite d'algorithmes paralleles classiques, dument repertories et consideres comme references, ceci afin de pouvoir comparer nos resultats et notre modele a ce qui existe.
Les algorithmes presentes ici etant coherents avec le modele formel *A* precedemment decrit, nous procedons a quelques rappels et precisions qui nous ont parus necessaires.

1.1. Rappels:

L'algorithme *A* formel, comporte un reseau G de p processus. Ce reseau est predetermine, donne a l'avance, il est impose par la structure physique du systeme informatique sur lequel il est destine a etre implemente. De la structure de ce reseau depend en particulier le mode d'acces aux donnees, et le fait qu'elles soient locales ou globales (i.e. partagees), donc en dernier ressort, la nature de l'algorithme, parallele ou distribue. Nous avons vu precedemment, que, pour ce qui concerne l'etude des complexites et efficacite de tels algorithmes, on pouvait construire un modele formel unique, la distinction entre les deux grandes familles d'algorithmes pouvant, pour notre propos, se ramener a une particularisation de la topologie du reseau de communication entre les processus, clique ou etoile dans le cas parallele, quelconque (mais connexe bien sur) dans le cas distribue.

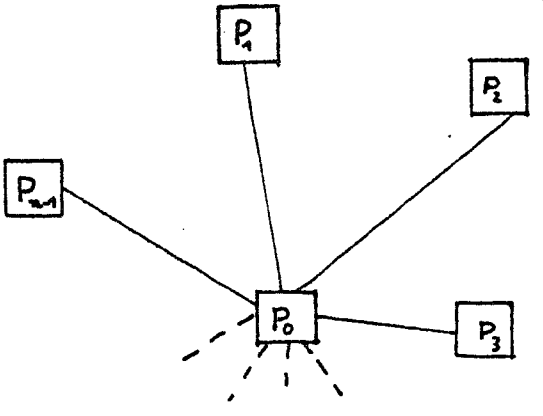
Nous allons donc par consequent examiner les deux cas:

a) **Cas parallele:** Le graphe G representatif des liaisons possibles entre les processus est soit une clique, soit une etoile (on considere la memoire comme un processus particulier). Afin

de minimiser le cout des communications on considere, construit sur G, un arbre couvrant de hauteur 1 dont la racine est la memoire; on obtient ainsi un m-arbre (m = p-1) - ou arbre m-aire, selon la terminologie de Knuth [KNU.] - couvrant dans lequel chaque processus est dote d'une memoire propre, locale (ou consideree comme telle) (voir fig. 1).

Comme specifie dans le modele de l'algorithme *A*, le calcul est considere comme ayant demarre des lors que l'un au moins des processus a demarre, et il s'arrete lorsque tous les processus sont arretes au bout d'un temps fini, ce qui se produit, lorsque, a la racine de l'arbre, le resultat est identifie. Tous les processus contiennent le meme code de programme (en fait, il ne s'agit pas la d'une obligation, mais la "specialisation" d'un processus est susceptible de poser plus de problemes qu'elle n'en resoud, de plus elle reintroduit souvent, explicitement ou implicitement une synchronisation, elle peut aussi poser des problemes dans la gestion des communications, generant des goulets d'etranglement). La complexite en temps de l'algorithme (en fonction de la taille du probleme) est le temps ecoule, ou nombre de pas (ou de transitions dans l'espace des etats) de calcul effectues entre le demarrage du premier processus, et l'arret du dernier. Durant les etapes du calcul, les p processus travaillent et envoient leurs resultats vers la racine qui les redistribue immediatement de facon a toujours avoir un maximum de processus actifs durant tout le deroulement de l'algorithme.

figure 1 etoile

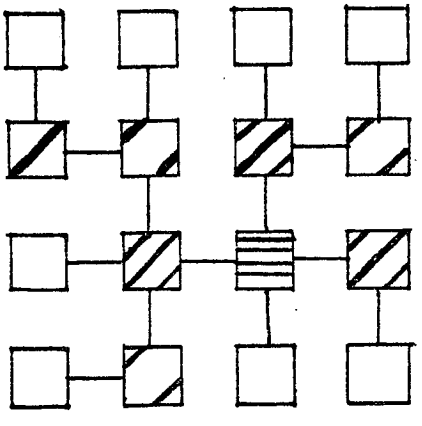


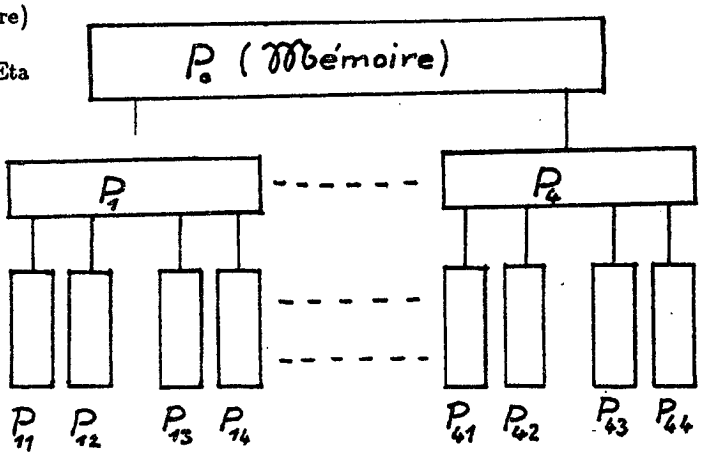
Quelle que soit la taille du reseau, les communications sont reduites a une constante, independante de la taille du probleme, representant les aller-retours des messages des p-1 processus-feuilles, a la racine de l'arbre. Les delais de communication dans ce cas sont de complexite en temps $O(1)$ au total.

b) **Cas distribue** Pour un reseau distribue, le graphe n'est pas complet, seules certaines liaisons de communication sont donnees entre processus (reseau d'interconnection). Dans ce cas, on peut soit construire un arbre couvrant de hauteur minimale [1] soit considerer que cet arbre est donne par la structure du reseau. C'est le cas deja pour les machines du type CDC Cyber Eta ou le modele de PRAM est abandonne au profit d'une structure en arbre. La structure est inherente a l'architecture de la machine ou du reseau considere.(voir fig. 2)

Une architecture de machine analogue, (voir [WA,MA.]) ainsi basee sur une structure en grappes disposees en arbre a ete proposee pour resoudre des problemes a tres forte complexite comme les problemes d'enumeration implicite du type voyageur de commerce, recherche

figures 2 (reseau en clique ou cellulaire) avec sa correspondance en arbre.
figure 3 structure en grappe du cyber Eta





[1] Le probleme de l'arbre couvrant de diametre borne pour un graphe non value (i.e. dont toutes les aretes sont equiponderees) est *polynomial* en temps (cf. [GA,JO.]).

d'ensemble absorbant minimal dans un graphe, etc... (ces problemes sont "NP-complets" dans le cas sequentiel). On peut considerer la "structure en grappe" comme un arbre m-aire, ou m-arbre. On prend p (p est le nombre de processus), par convention, comme etant une puissance de deux: $p = 2^k$. Evidemment, dans une telle structure, m doit etre le plus grand possible. Pour un m-arbre complet, on a:

Si h est la hauteur du m-arbre *parfait* associe; le nombre de sommets a hauteur 1, 2, 3, ... , h est respectivement $m, m^2, m^3, \cdots, m^h$; le nombre de feuilles etant alors $p = m^h$ ($p = 2^k$ par ailleurs) .

Donc, pour un m-arbre complet associe a l'arbre parfait de hauteur h, $h \leq \lceil \log_m p \rceil$

Si m est assez grand, h s'approche de loglog p en fonction du nombre de processus p. (Le tableau en annexe met en correspondance les valeurs de m et de loglog p pour un certain nombre de valeurs de p differentes ($m = e^{\frac{\ln p}{\log\log p}}$) [2]. Il permet de voir que $h = \bigcirc(\log_m p)$ s'approche de $\bigcirc(\log\log p)$ pour des valeurs de m tout a fait acceptables.)

Donc, si G n'est pas une clique, les delais de communication entre les p processus sont toujours de l'ordre de $\log_m p$, puisque de l'ordre de h = hauteur du m-arbre parfait asssocie a G. Pour m convenablement choisi, ces delais sont donc exprimables en $\bigcirc(\log\log p)$.

C'est un resultat tres appreciable eu egard a ceux obtenus sur d'autres modeles. Par exemple, [PA,KO,RO.], ou, dans un autre modele de reseau de communications entre processus, un anneau (uni ou bi-directionnel), Pachl, Korach et Rotem trouvent un nombre de messages, en moyenne, pour le probleme de recherche du maximum de l'ordre de $\Omega\ (p \cdot \log p)$ pour p processus, et, dans le pire des cas, de l'ordre de $1{,}89\ p.\log p + \bigcirc(p)$.

Par ailleurs, la mesure de complexite C(N, p) des algorithmes (N etant la taille du probleme) est :

$$\text{Max } \{ \text{delais des communications} + T(N, p) \},$$

[2] Lorsque la base des logarithmes n'est pas precisee, elle est 2; par ailleurs, la notation *ln* signifie logarithme neperien.

Ou T(N, p) represente la complexite en temps des seuls calculs (ou nombre de pas de calcul necessaires pour un probleme de taille N, au processus qui effectue le plus de calculs). On aura donc, lorsque G n'est pas une clique :

$$C(N, p) = \max \{ O(\log\log p) + T(N, p) \}$$

Dans les problemes de calcul du maximum de N elements, de fusion de deux listes de tailles respectives M et N, et de tri de N elements, notre modele permet d'atteindre les bornes optimales de Valiant [VAL.] et Kruskal [KRU.] a une constante pres dans presque tous les cas, en tenant compte des differents phenomenes susceptibles d'oberer les performances de l'algorithme, en particulier les delais de communication entre les processus et sans probleme de conflit d'acces memoire. Pour ce faire, nous montrons comment implementer sur notre modele les algorithmes de Valiant essentiellement, ameliores par Preparata et generalises par Kruskal.
Si nous prenons pour base de travail et de reference ces algorithmes, c'est essentiellement parce qu'ils atteignent des *bornes optimales* (dans le pire des cas) et qu'on peut facilement les implementer sur notre modele, comme nous le verrons. Par ailleurs, les algorithmes de Valiant [VAL.] representent effectivement une classe d'algorithmes "optimaux" (pour les problemes ou dominent les comparaisons) permettant d'exploiter a fond les possibilites du parallelisme - comme le souligne Valiant lui-meme.
Son algorithme de fusion en particulier est, de par ses performances en "temps de calcul", un moule ideal pour construire de tres bons algorithmes de tri (cf. [KRU.]). D'excellents resultats asymptotiques sur les tris, comme ceux de [A,K,S.] qui realisent des performances en temps $O(\log n)$ avec n processeurs, sont moins interessants pour nous, car, outre le fait que leurs constantes sont tres grandes, leurs algorithmes semblent moins generaux et plus compliques - de meme d'ailleurs que leur modele.

- Dans le cas ou G est une clique,

toutes les bornes optimales sont atteintes pour les algorithmes de [VAL.] et [KRU.], compte tenu des delais de communication (contrairement a Valiant qui n'en tient pas compte).

- Dans le cas ou G n'est pas une clique,

pour $p > N$ les bornes pour trouver le maximum de N elements calculees ne sont pas atteintes, non plus que celles calculees par Kruskal pour la fusion de deux listes lorsque $p \geq \lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$.

Ceci est bien comprehensible puisque les bornes superieures de Valiant et Kruskal sont legerement pejorees par les delais de communication dans le cas ou p est superieur a la taille du probleme ($p > N$): ces delais restent en effet quasi-identiques a p fixe, les performances en nombre de comparaisons devenant au contraire bien meilleures lorsque le nombre de processus depasse la taille du probleme.

Cependant, pour des tailles de problemes N deja grandes vis a vis de p, nous verrons que les meilleures bornes sont bien atteintes pour notre modele.

Pour faire un parallele avec les travaux de Shiloach et Vishkin [SH,VI.], leur modele semble moins realiste et moins performant puisque les algorithmes de fusion et de tri de Valiant, qui sont optimaux, ne peuvent etre implementes sur leur modele du fait des problemes d'allocation de processus: allocation de p processus a p taches en temps constant.

Nous obtenons les resultats suivants:

1.2. Identification du maximum de N elements dans le pire des cas.

G etant le graphe des communications entre les processus:

1.2.1. Si G est une clique:

tous les delais de communication etant en $\bigcirc(1)$ dans l'arbre couvrant de hauteur 1, sur G, les bornes obtenues par Valiant sont atteintes:

$$\underset{p}{Max}(N) = \odot(\frac{N}{p} + loglogp)$$

si $p \leq N$; (optimal pour $p \leq \frac{N}{loglogN}$).

$$\underset{p}{Max}(N) = \bigcirc(loglog N - loglog(\frac{p}{N} + 1))$$

si $N \leq p \leq \binom{N}{2}$

et donc,

$$\underset{p}{Max}(N) = \bigcirc(logk)$$

si $p = \lfloor N^{1+\frac{1}{k}} \rfloor$, $(k > 1)$.

1.2.2. Si G n'est pas une clique.

$$\underset{p}{Max}(N) = \bigcirc(\frac{N}{p} + 2loglogp) \quad si \ p \leq N$$

$$\underset{p}{Max}(N) = \bigcirc(loglogp + loglog\ N - loglog\ \frac{p}{N}) \quad si N \leq p \leq \binom{N}{2}$$

En effet, dans ce cas, la structure de multiprocesseurs "en grappes" induit des delais de communication dans le pire des cas en $\bigcirc(log_m p)$ et, en choisissant correctement m, on atteint facilement $\bigcirc$(loglog p) pour p fixe. Dans le cas ou $N \leq p$, le resultat de valiant:

$$loglog\ N - loglog\ \frac{p}{N} - \text{Constante} \leq \underset{p}{Max}(N) \leq loglog\ N - loglog\ \frac{p}{N} + \text{Constante}$$

qui ne tient compte que des comparaisons effectuees ne peut etre atteint; c'est donc bien la borne liee aux delais de communication, en $\bigcirc(\log_m p)$ qui pejore la mesure de complexite.

1.3. Fusion de deux listes de tailles N et M $(N \leq M)$.

Toujours dans le pire des cas, en notant $Fusion_p$ la complexite de l'algorithme.
G etant le graphe des communications entre les processus.

1.3.1. Si G est une clique:

$$Fusion_p(N,M) \leq \frac{k}{\log k} \cdot loglog\ N + k + 1 + \bigcirc(1) ,$$

pour

$$p = \lfloor N^{1-\frac{1}{k}} \cdot M^{\frac{1}{k}} \rfloor ; \quad k \geq 2; 2 \leq N \leq M$$

On atteint l'optimalite pour k = 3. Alors, $p = \lfloor N^{\frac{2}{3}} \cdot M^{\frac{1}{3}} \rfloor$, et on obtient :

$$Fusion_p(N,M) = 1{,}893\ loglog\ N + 4 + \bigcirc(1)$$

De meme,

$$Fusion_p(N,M) \leq \frac{k}{\log k} \cdot (loglog\ N - loglog\ r) + k + 1 + O(1)$$

pour $p = \lfloor r \cdot N^{1-\frac{1}{k}} \cdot M^{\frac{1}{k}} \rfloor$ ($k \geq 2$; $2 \leq r \leq N \leq M$). L'optimalite est encore obtenue pour k = 3;

$$Fusion_p(N,M) \leq \frac{M+N}{p} + \log\left(\frac{M+N}{p}\right) + \frac{3}{\log 3} \cdot loglog\ p + 6\ O(1)$$

si $2 \leq p \leq M + N$ et k = 3 (cas optimal)

1.3.2. Si G n'est pas une clique:

Alors, pour $p = \lfloor N^{1-\frac{1}{k}} \cdot M^{\frac{1}{k}} \rfloor$ avec $2 \leq N \leq M$,

$$Fusion_p(N,M) = O(loglog\ p + \frac{k}{logk} loglog\ N + k + 1)$$

et pour $p = \lfloor r \cdot N^{1-\frac{1}{k}} \cdot M^{\frac{1}{k}} \rfloor$ avec $k \geq 2$, et $2 \leq r \leq N \leq M$

$Fusion_p(N,M) = O(loglog + \frac{k}{logk} \cdot (loglog\ N - loglog\ r) + k + 1)$.

Et pour p tel que $2 \leq p \leq N + M$, on a :

$$Fusion_p(N,M) \leq \frac{M+N}{p} + \log\left(\frac{N+M}{p}\right) + \frac{3 + \log 3}{\log 3} loglog\ p + 6 ,$$

ceci, dans le cas optimal.

1.4. Tri de N elements .

G etant le reseau de communication entre les processus.

1.4.1. Si G est une clique:

Comme precedemment, tout delai s'exprime en $O(1)$ independamment de N et de p, et les resultats optimaux de Kruskal generalisant ceux de Preparata sont atteints:

Si $p = N$

$$Tri_p \leq 1{,}893 \cdot \frac{\log N \cdot loglog\ N}{logloglog\ N} \cdot (1 + O(\frac{logloglogloglog\ N}{logloglog\ N}))$$

Si $N \geq \frac{3p}{2ln3} \cdot loglog\ p$

$$Tri_p(N) \leq \frac{N}{p} \cdot \log N + \frac{3log\ p}{\log 3} \cdot loglog\ p + O(\frac{N}{p} + \log p \cdot \log \frac{2N}{p})$$

Si $p = N \cdot (log N)^{\frac{1}{k}}$ $(k \geq 2)$

$$Tri_p(N.) \leq 1{,}893 \cdot k \cdot \log N + O(k \cdot \log N) .$$

1.4.2. Si G n'est pas une clique:

Dans ce cas, les delais sont de l'ordre de $loglog\ p$ et on a par consequent:

$$Tri_p(N) \leq 1{,}893 \cdot \frac{\log N \cdot loglog\ N}{logloglog} N \cdot (1 + O(\frac{logloglogloglog\ N}{logloglog} N + loglog N))$$

Si $N \geq \frac{3p}{2ln3} \cdot loglog\ p$

$$Tri_p(N) \leq \frac{N}{p} \cdot \log N + \frac{3log\ p}{log3} \cdot loglog\ p + O(\frac{N}{p} + \log p \cdot \log \frac{2N}{p} + loglog\ p)$$

Si $p = N \cdot (\log N)^{\frac{1}{k}}$ avec $k \geq 2$

$$Tri_p(N) \leq 1{,}893 \cdot k \cdot \log N + O(k \cdot \log N + loglog\ p).$$

En particulier, on peut trier dans ce cas N nombres avec $p = N(log\ N)^{\frac{1}{k}}$, en temps total $O(k.\log N)$ dans le pire des cas. Ce resultat ameliore les valeurs obtenues pour les complexites des meilleurs tris implementes sur le modele de Shiloach et Vishkin [SH,VI.].

(i.e. : $Tri_p(N.) = O(k.logN)$ si $p = \lceil N^{1+\frac{1}{k}} \rceil$, $(k > 1)$).

Par ailleurs, comme le precise lui meme Valiant, les algorithmes utilises ne tiennent compte que des comparaisons (les temps de transmission entre processus etant mis a part) et ce, dans le pire des cas. Mais, toute borne inferieure sur le temps de calcul dans ce modele parallele est aussi une borne pour tout modele parallele dont les operations de base sont des comparaisons deux a deux entre elements. Les trois algorithmes presentes ont donc un caractere de grande generalite puisque, lorsque le temps de calcul necessaire aux comparaisons est le facteur dominant, la meilleure borne superieure constructive correspond a celle de l'algorithme

le plus rapide pour des machines multiprocesseurs.

Lorsque ce sont les delais dus aux communications qui sont dominants, le modele *A* que nous proposons induit une complexite dans le pire des cas en $O(loglog\ p)$ qui se rapproche des solutions optimales realistes.

2. RECHERCHE DU MAXIMUM.

Nous allons etudier l'evolution des bornes des performances de l'algorithme de Valiant, implemente sur notre modele, en fonction des hypotheses faites sur le nombre de processus disponibles.

2.1. L'algorithme de Valiant et son implementation:

Il s'agit ici de trouver le maximum parmi N elements $a_1, a_2, \cdots, a_N$.

Shiloach et Vishkin [SH,VI.] ont implemente l'algorithme de Valiant sur un modele de WRAM [3] et ont atteint les bornes calculees par Valiant. Pour l'implementation de l'algorithme sur notre modele, nous reprenons les idees fondamentales de [SH,VI.], en les adaptant.

Nous allons decrire brievement l'algorithme de Valiant et examiner les problemes d'implementation qui y sont lies.

2.1.1. L'algorithme de Valiant.

On suppose que $a_i \quad a_j$ pour tout $i \quad j$. Si $a_i = a_j$ avec $i < j$, alors, a_i est considere comme etant "plus grand" que a_j. (Cette stabilite est surtout importante dans le cas de l'algorithme de fusion).

En resume, l'algorithme consiste a trouver un maximum parmi les elements en utilisant $\binom{k}{2}$ processus; le modele propose par Valiant [Val.] realise ceci en une seule comparaison. Chaque processus compare une paire differente (a_i, a_j) et toutes les informations etant a sa disposition, il trouve l'element maximal sans autre comparaison.

---

[3] Une WRAM restreint les possibilites d'ecriture simultanee de deux processeurs en memoire partageable au cas ou il y a identite de contenu des messages destines a une meme cellule de memoire. Une CREW-PRAM (Concurrent Read, Exclusive Write PRAM) interdit toute ecriture simultanee en memoire commune (cf. [BLU.] et [BO,HO.]).

Dans l'analyse de Valiant, ni les problemes d'allocation de processus aux paires d'elements a comparer, ni les delais necessaires aux communications ne sont pris en compte.

L'etape iterative de l'algorithme est la suivante:

On realise des comparaisons uniquement entre des elements "candidats" (i.e. qui n'ont encore jamais "perdu" dans aucune comparaison anterieure). Ainsi, les candidats a l'etape t sont compares entre eux a l'etape t+ 1.

Chacune des etapes est donc decomposee en trois parties:

a) L'ensemble des elements en entree est partitionne en un nombre **minimal** d'ensembles $S_1, S_2, \cdots, S_l$ tels que:

(1) $\| |S_i| - |S_j| \| \leq 1 \quad (\forall\ 1 \leq i < j \leq l)$ avec $|S_i| = Card\ S_i$

(2) $\sum_{i=1}^{i=l} \binom{|S_i|}{2} \leq p$ p etant le nombre de processus (c'est l'ordre du graphe G).

b) A chaque ensemble S_i est assigne un nombre de processus $\binom{|S_i|}{2}$ qui, bien sur, est suffisant pour trouver le maximum, appelons le a^i, en un nombre de pas de comparaisons constant a chaque etape (et non en un pas *unique* de comparaison comme dans [VAL.], ce qui est impossible a realiser sur notre modele).

c) Si $l = 1$, *le maximum est trouve;*

sinon $a^1, a^2, \cdots, a^l$ *represente l'ensemble des candidats a l'etape suivante.*

Valiant montre que pour p = N, il suffit de loglog N + Const. etapes pour arriver au maximum, et que ceci represente une borne inferieure pour ce cas.

Les differents problemes a resoudre pour l'implementation de cet algorithme sont:

i) allocation des processus aux ensembles de candidats, en temps constant;

ii) calcul de l, entier minimal verifiant (1) et (2);

iii) a chaque etape, calcul du maximum en un nombre constant de comparaisons (ce nombre allant en diminuant a chaque etape).

Notre implementation resoud ces trois difficultes, ce qui permet a l'algorithme de conserver la meme complexite que celui de Valiant, ... a une constante pres; cette constante pro-

vient du nombre constant de comparaisons a effectuer a chaque etape (en prenant le maximum de ces nombres de comparaisons sur l'ensemble des etapes, on obtient bien les memes bornes, avec des constantes differentes).

2.2. L'algorithme et le modele:

On trouvera l'algorithme proprement dit dans [SH,VI.]. Nous allons, dans ce paragraphe discuter cet algorithme du point de vue des difficultes d'implementation qui y sont inherentes. De ce point de vue, il faut remarquer que notre modele n'engendre *aucun* conflit de recherche en memoire commune, contrairement a ce qui peut se produire dans une WRAM ou une CREW-PRAM par exemple, puisque ici chaque processus connait l'ensemble du programme et que chaque processus a sa propre memoire locale. Aucun conflit d'acces a la racine (dans le cas d'une clique) ou a un controleur intermediaire (bus + memoire comme dans [WA,MA.]), dans le cas d'un graphe structure en "grappes" ne peut survenir.

Dans la description de l'algorithme, on suppose d'abord $N \leq p$.

- Le premier probleme que nous allons examiner est celui du calcul de l.

Soit: $n \ mod \ l = n - l \ . \lfloor \frac{n}{l} \rfloor$, l'inegalite (1) sur les ensembles S_i , S_j $(1 \leq j \leq i \leq l)$ s'explique par le fait que l'ensemble des candidats *a une etape quelconque* est partitionne en $n \ mod \ l$ sous ensembles de taille $\lfloor \frac{n}{l} \rfloor + 1 = \lceil \frac{n}{l} \rceil$ et $l - n \ mod \ l$ sous ensembles de taille $\lfloor \frac{n}{l} \rfloor$.

l est alors le plus petit entier verifiant (2), c'est a dire:

$$(3) \qquad n \ mod \ l \ . \binom{\lceil \frac{n}{l} \rceil}{2} + (\ l - n \ mod \ l \ ) \ . \binom{\lfloor \frac{n}{l} \rfloor}{2} \leq p$$

avec:

p nombre de processus dans G (donnee fixe au cours du calcul)

n taille de l'ensemble des candidats a une etape quelconque. Au debut du calcul, bien sur n = N = taille de la liste totale dont on cherche le maximum.

$n = N = Card.(\ L \ )$ ou $L = \{a_1, a_2, \cdots , a_N\}$. A l'etape suivante,

$n = l_1$ est la taille de la liste $L_1 = \{a^1, a^2, \cdots , a^{l_1}\}$,

l est defini au depart comme etant le plus petit entier tel que L est partitionne en

N *mod* l sous ensembles de taille $\lceil \frac{N}{l} \rceil$, et

l − N *mod* l sous-ensembles de taille $\lfloor \frac{N}{l} \rfloor$ et alors, on a a la premiere etape:

$$N \bmod l \cdot \binom{\lceil \frac{N}{l} \rceil}{2} + (l - N \bmod l) \cdot \binom{\lfloor \frac{N}{l} \rfloor}{2} \leq p.$$

A l'etape suivante, les l premiers candidats ayant ete degages par la premiere etape, on doit calculer l_1 verifiant (3) avec $n = l < N$ et on obtient $l_1 < l$, etc d'etape en etape, jusqu'a l'etape r *finale* ou on a $l_r = 1$ qui donne le maximum de la liste initiale.

On calcule facilement l a chaque etape en un nombre constant d'operations (independant de N et p) par la methode utilisee par [SH,VI.]:

n et l variant d'une etape a la suivante, ainsi que L, on les exprime en fonction du numero r de l'etape consideree. Ainsi, L[r, j] est un tableau pour lequel a(r,j) est le j-ieme element dans l'ensemble des candidats a l'etape (r+ 1).

On cree ainsi un vecteur A = (0, 0, ... , 0, ... , 1, 1, ... , 1) en placant un 1 en a(i) quand c'est possible, et 0 sinon. Ainsi, le l cherche est le premier indice de A pour lequel a(i) = 1. (Si $p_i > N$, le processus p_i reste inactif).

- Le second probleme que nous examinons est celui de la selection des candidats a chaque etape, en un nombre *constant* de comparaisons.

Supposons qu'a l'etape 1, le calcul de l nous conduise a subdiviser L, ensemble a N elements, donne au depart, en k sous-ensembles de taille n et k' sous-ensembles de taille n' (avec evidemment kn + k'n' = N, k + k' = l, et n, n' < N). Les k sous-ensembles de taille n sont assignes a $\binom{n}{2} = \frac{n' \cdot (n'-1)}{2}$ processus chacun, avec $k \binom{N}{2} + k' \binom{n}{2} \leq p$.

C'est exactement le nombre necessaire pour effectuer toutes les comparaisons necessaires 2 a 2 entre elements dans les l sous-ensembles ainsi definis.

Dans les k sous-ensembles de taille n affectes a $\binom{N}{2}$ processus chacun, les comparaisons s'effectuent en parallele de la meme maniere: les n elements sont compares deux a deux par chacun des $\binom{N}{2}$ processus. Il y a ainsi $\binom{N}{2}$ paires d'elements distinctes comparees, et le nombre de pas de comparaisons necessaires pour trouver le maximum verifie les equations

recurrentes:

$$C_n = C_{\lceil \frac{n}{2} \rceil}$$

et

$$C_1 = 0$$

Ce qui donne le resultat $C_n = \lceil \log_2 n + 1 \rceil$.

De meme, avec les k' sous-ensembles de taille n', on obtient $C_{n'} = \lceil \log_2 n' + 1 \rceil$ et le nombre de comparaisons a effectuer en parallele durant cette premiere etape est:

$$C_N = Max \{ C_n , C_{n'} \} = Max \{ \lceil \log n + 1 \rceil , \lceil \log n' + 1 \rceil \}.$$

Il est clair que C_N est superieur aux C_n des autres etapes et donc, sur l'ensemble de l'execution de l'algorithme, toutes etapes prises en compte, le maximum est trouve en un nombre de pas de comparaison effectuees sur les candidats qui sera au plus de $C_N = \underset{1 \le i \le l}{Max} \{ \lceil \log | S_i | + 1 \rceil \}$

- Le dernier probleme que nous examinerons est celui de l'allocation des processus (il y en a p dans G) aux donnees du probleme. (Ce probleme a deja ete partiellement aborde dans les paragraphes precedents).

Pour le calcul de l, chaque processus effectue le test precedemment decrit et aucune difficulte particuliere ne peut surgir. De meme a chaque etape, les l_r (r numero de l'etape) candidats sont affectes par paires au nombre de processus necessaires. On alloue un processus a une paire donnee, directement, en prenant seulement garde a ce que *tous* les l_r candidats soient presents au moins dans une paire ainsi affectee. (Ceci est fait par l'algorithme sans difficulte particuliere, en prenant les elements en question, l'un apres l'autre, deux a deux).

La reallocation des processus pour la comparaison des candidats obtenus apres la premiere comparaison dans les sous-ensembles se fait automatiquement puisque seuls sont gardes les elements les plus grands de chaque couple initial, et ainsi de suite jusqu'aux maximas des l_r sous-ensembles.

2.3. Complexites et resultats

A chaque etape, les maximas sont obtenus en un nombre constant d'operations (independant de N ou p).

De [VAL.] on tire que le nombre d'etapes est au plus de loglog N + 1 .

Corollaire 1: [VAL.]

$$\underset{p}{Max}(N) = \bigcirc(loglog\ N - loglog(\frac{p}{N} + 1)) \quad \text{pour } N \leq p \leq \binom{N}{2} \quad \text{si G est une clique.}$$

et

$$\underset{p}{Max}(N) = \bigcirc(loglog\ p + loglog\ N - loglog\ \frac{p}{N}) ,$$

si G n'est pas une clique, puisqu'alors les delais en $\bigcirc$(loglog p) depassent les comparaisons a effectuer

(en $\bigcirc$(loglog N)).

Corollaire 2 : [VAL.]

Si $1 < p \leq N$, si G est une clique,

$$\underset{p}{Max}(N) = \bigcirc(\frac{N}{p} + loglog\ p)$$

et

$$\underset{p}{Max}(N.) = \bigcirc(\frac{N}{p} + 2loglog\ p) ,$$

si G n'est pas une clique.

PREUVE :[VAL.] On partitionne d'abord les N elements en p ensembles disjoints de tailles $\lceil \frac{N}{p} \rceil$ ou $\lfloor \frac{N}{p} \rfloor$. On alloue chaque processus a un ensemble, et on trouve le maximum sequentiellement. Trouver le maximum parmi les p selectionnes ainsi obtenus se fait par l'algorithme decrit ci-dessus.

Corollaire 3:

L'algorithme est optimal pour $1 < p <= \frac{N}{loglog N}$ pour tout G.

On a alors $\underset{p}{Max}(N) = \bigcirc(\frac{N}{p})$.

THEOREME : [VAL.] Si G est une clique,

$$\underset{p}{Max}(N) \geq loglog\ N - loglog(\frac{p}{N} + 1) \text{ pour}$$

$N \leq p \leq \binom{N}{2}$

En particulier, $\underset{p}{Max}(N) = \bigcirc(\log k)$ si $p = \lceil N^{1+\frac{1}{k}} \rceil \quad (k > 1)$.

Si G n'est pas une clique, cette borne n'est pas optimale et on a

$$\underset{p}{Max}(N) \geq loglog\ p + loglog\ N - loglog(\frac{p}{N} + 1) \text{ avec } N \leq p \leq \binom{N}{2}$$

PREUVE: Valiant demontre que la borne inferieure est optimale meme si l'on ne compte que les comparaisons. Donc, dans le cas ou G n'est pas une clique, les delais de communication l'emportent (sur le resultat de Valiant) et dans ce cas,

$$\underset{p}{Max}(N) \geq loglog\ p + loglog\ N - loglog\ (\frac{p}{N} + 1);\ (N \leq p).$$

3. FUSION DE DEUX LISTES TRIEES.

Nous allons examiner ici, l'algorithme de Valiant et Kruskal, et son implementation , et les consequences qui en decoulent sur notre modele

3.1. L'algorithme de Valiant et Kruskal et son implementation:

Borodin et Hopcroft [BO,HO.] ont montre que l'algorithme de fusion de deux listes triees (de, respectivement N et M elements - $N \leq M$ -) propose par Valiant pouvait etre implemente sur une CREW-PRAM en conservant le meme cout en nombre de pas de comparaison que le resultat de Valiant - a une constante pres, et ce, malgre de nombreuses difficultes dans cette implementation. En particulier le *probleme de l'allocation des processeurs* qui *n'est pas resolu* par Shiloach et Vishkin [SH,VI.] sur une WRAM.

Kruskal ameliore et generalise l'algorithme de Valiant. L'implementation se fait sur une CREW-PRAM [KRU.] selon la demonstration de [BO,HO.].

C'est ce dernier algorithme,optimal a une constante pres pour $k = 3$ et $M = N$, que nous allons utiliser ici.

Il s'agit donc de fusionner une liste de N elements avec une autre de M elements ($N \leq M$).

La demonstration de l'algorithme se fait par induction. On montre que, etant donne un reseau de p processus, avec $p = \lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$, avec k entier ≥ 2 , il est possible de reduire le probleme initial (de taille $N \leq M$) en k etapes de comparaison, au probleme suivant: fusionner un certain nombre de couples de listes triees telles que, dans chaque couple, la liste la plus petite soit de taille inferieure a $N^{\frac{1}{k}}$. Les couples de listes sont construits de telle sorte que les $p = \lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$ processus leur sont alloues en s'assurant que, pour chaque paire, le nombre de processus est suffisant pour satisfaire l'hypothese d'induction.

Soit l'algorithme suivant pour les listes triees $X = (x_1, x_2, \cdots, x_N)$ et $Y = (y_1, y_2, \cdots, y_M)$, (cf. [VAL.] pour k= 2).

-a) On marque les elements de la liste X d'indices $i . \lceil N^{1-\frac{1}{k}} \rceil$ et ceux de la liste Y d'indices $i . \lceil M^{\frac{1}{k}} \rceil$ pour i = 1, 2, ...

Il y en a au plus respectivement $\lfloor N^{1-\frac{1}{k}} \rfloor$ et $\lfloor M^{\frac{1}{k}} \rfloor$. Les sous listes de X et Y ainsi creees entre deux elements marques successifs et apres le dernier element marque dans chaque liste sont appeles des "segments" (en nombre $\lfloor N^{1-\frac{1}{k}} \rfloor + 1$ et $\lfloor M^{\frac{1}{k}} \rfloor + 1$ respectivement).

- b) On compare chaque element marque de X a chaque element marque de Y. Ceci necessite $\lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$ comparaisons, ce qui peut etre fait en une unite de temps - une etape de comparaison - avec les p processus.

- c) A partir des comparaisons de (b), on peut decider pour chaque element marque, quel est le segment de l'autre liste dans lequel on doit l'inserer. Ensuite, on peut comparer chaque element marque de X a chaque element du segment de Y qui lui correspond (on trie ces segments comprenant les elements marques de X correspondants). Cette operation demande au plus

$$\lfloor N^{1-\frac{1}{k}} \rfloor . (\lceil M^{\frac{1}{k}} \rceil - 1) < \lceil N^{1-\frac{1}{k}} . M^{\frac{1}{k}}$$

comparaisons en tout, et peut donc s'effectuer en une unique unite de temps avec p processus.

Apres (a), (b), et (c), en 2 etapes de comparaisons, on a identifie l'emplacement de chaque element marque de X dans Y. Il ne reste plus qu'a fusionner les paires disjointes de sous-listes (X_i , Y_j) ou X_i est un segment de X, et Y_i un segment de Y. Donc, $|X_i| \leq N^{\frac{1}{k}}$ et $\sum_i |X_i| < N$, $\sum_i |Y_i| < M$ ($\forall\ i$) , puisque ces sous listes sont disjointes.

Le processus inductif transformant une paire de listes en une paire de sous-listes peut ainsi continuer avec p processus, et la taille du plus petit sous-ensemble de chaque paire de sous listes est bornee inductivement par la puissance $\frac{1}{k}$ ieme de la plus petite composante de la paire (on demontre ainsi le theoreme 1 - voir partie 8.3.).

Les problemes d'implementation de cet algorithme se ramenent:

i) au marquage initial des elements d'indices $i \,.\lceil N^{1-\frac{1}{k}} \rceil$ et $i \,.\lceil M^{\frac{1}{k}} \rceil$ avec $N \leq M$, de X et Y respectivement,

ii) a la reallocation des processus (apres l'etape (c)) aux paires de sous listes disjointes restant a fusionner.

Un autre probleme reste non-resolu pour un modele de type WRAM (cf. [SH,VI.] et [PRE.]) et est tres complique a resoudre sur une CREW-PRAM (cf. [BO,HO.]), c'est la reallocation.

Pour notre modele, au contraire, la reallocation se fait naturellement dans la mesure ou tous les processus ont connaissance du programme dans son ensemble, et possedent chacun une memoire locale. Les seules "limitations" sont celles des delais de transmission entre processus directement pour une clique, par passage par un processus controlleur (bus + memoire comme dans le Cyber Eta ou dans le modele propose par [WA,MA.]) dans le cas d'un reseau "en grappes".

3.2. L'algorithme et le modele:

L'enonce de cet algorithme de fusion de deux listes triees est donne dans [PRE.].

Le probleme (i) de marquage initial se resoud tres simplement en implementant les listes X et Y sous une forme un peu particuliere, a savoir un tableau a $\lceil N^{1-\frac{1}{k}} \rceil$ lignes pour X, et $\lceil M^{\frac{1}{k}} \rceil$ lignes pour Y (liste X en $\lceil N^{1-\frac{1}{k}} \rceil$-representation et Y en $\lceil M^{\frac{1}{k}} \rceil$-representation). Ainsi, les elements marques de X et de Y sont, dans cette representation, les elements des dernieres lignes des tableaux X et Y respectivement.

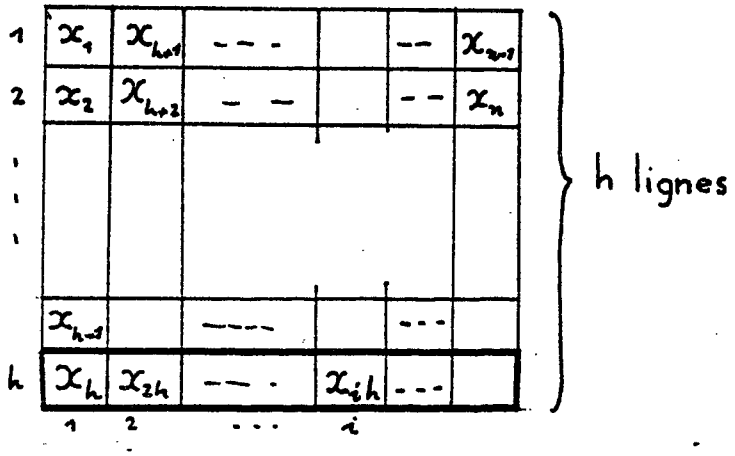
EXEMPLE $X = (x_1, x_2, \cdots, x_n)$. On marque tous les h elements de X, avec $h = \lceil N^{1-\frac{1}{k}} \rceil$. X peut donc se representer sous forme de h lignes comme dans la figure, et les elements marques d'indices de type i.h, (i= 1,2,3,...) sont dans la derniere ligne.

Les segments etant des sous listes de X et Y creees par les elements marques de X et de Y, ils apparaissent dans cette representation comme les (h-1) premiers elements de chaque colonne differente de la derniere colonne et, dans celle-ci, comme les elements de toute la derniere colonne si N k . h ou des (h-1) premiers elements de la derniere colonne si N = k . h ($k \in \mathbb{N}$) dans ce dernier cas les segments sont tous de meme taille.

---

h-représentation de $X=(x_1, x_2, \cdots, x_n)$

avec $h = \lceil N^{1-1/k} \rceil$



La comparaison entre elements marques de X et Y consiste en l'allocation des dernieres lignes de X et de Y (comme ensembles d'elements) aux processus qui sont en nombre suffisant: $\lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$ pour realiser ce travail. L'insertion de chaque element de la derniere ligne de la h-representation de X (respectivement de Y) dans le segment correspondant de Y (respectivement de X), puis les comparaisons de chaque element de la derniere ligne de X avec chaque element de son segment correspondant dans Y se font par allocation des processus adequats et mise en memoire "localement" ou "centralement" (dans la memoire affectee au "bus-racine" dans le cas d'un reseau "en grappes"). Dans le cas ou le reseau est une clique, le processus racine affecte les comparaisons a effectuer deux a deux aux autres processus qui lui transmettent les listes et sous listes fusionnees; celles-ci sont placees en memoire du processus racine jusqu'a ce que l'algorithme termine cette partie (c).

La reallocation des processus utilises dans les etapes (a), (b), (c) pour fusionner les sous listes restantes ne pose pas de probleme, qu'on ait un reseau de type clique, ou un reseau "en grappes", c'est a dire en structure d'arbre. Des qu'un processus est libere des taches de (a) et (b) et (c), on lui assigne les elements des sous listes non fusionnees...cette derniere fusion se fait en parallele sur tous les processus disponibles et ils sont en nombre suffisant pour l'effectuer. Les listes X et Y fusionnees sont dans la memoire du processus racine.

3.3. Complexite et resultats:

Suivant les resultats de Kruskal [KRU.],

Pour $p = \lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$ (k entier tel que, $k \geq 2$ et $2 \leq N \leq M$), on a:

$$Fusion_p(N,M) \leq \frac{k}{\log k} . loglog\ N + k + 1 \quad ([KRU.] \text{ Theoreme 2}).$$

L'optimum, atteint pour k = 3 donne :

$$Fusion_p(N,M) = 1{,}893 loglog\ N + 4 .$$

Nous obtenons alors les resultats suivants:

THEOREME 1:

Pour $p = \lfloor N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$ (r: { k entier } $\geq$ 2 et 2 $\leq$ N $\leq$ M)

- Si G est une clique

$$Fusion_p(N,M) \leq \frac{k}{\log k} . loglog\ N + k + 1 .$$

- Sinon

$$Fusion_p(N,M) \leq loglog\ p + \frac{k}{\log k} . loglog\ N + k + 1$$

car alors, $p > N$ et les delais de transmission en $loglog\ p$ sont donc superieurs au nombre de pas de comparaison.

PREUVE: [VAL.] et [KRU.]

Dans l'etape c), a l'instant k.i, chaque paire de sous-listes produite possede une composante de taille inferieure a $\lambda_i = \lfloor (\lambda_{i-1})^{\frac{1}{k}} \rfloor$ ($\lambda_0 = N$) .

Il ne reste plus qu'a resoudre la recurrence : $\lambda_i \leq N^{(\frac{1}{k})^i}$, i etant le plus petit entier tel que $\lambda_i > 0$.

Corollaire 1:

Pour $p = \lfloor N^{\frac{2}{3}} . M^{\frac{1}{3}} \rfloor$ (avec $2 \leq N < M$)

- Si G est une clique

$$Fusion_p(N,M) = 1{,}893 loglog\ N + 4$$

- Sinon

$$Fusion_p(N,M) \leq loglog\ p + 1{,}893 loglog\ N + 4$$

Corollaire 2:

Pour $p = \lfloor r . N^{1-\frac{1}{k}} . M^{\frac{1}{k}} \rfloor$, et $2 \leq r \leq N < M$.

- Si G est une clique

$$Fusion_p(N,M) \leq \frac{k}{\log k} .$$

- Sinon

$Fusion_p(N,M) \leq loglog\ p$

Corollaire 3 :

Pour $2 \leq p \leq M + N$, si G est une clique:

$$Fusion_p(N,M) \leq \frac{M+N}{p} + \frac{3}{\log 3} . loglog\ p + 6$$

et $Fusion_p(N,M) \leq \frac{M+N}{p} + \log(\frac{M+N}{p}) + \frac{3+\log 3}{log 3} loglog\ p + 6$ si G n'est pas une clique

PREUVE:

Semblable a celle de Valiant; il s'agit de generalisations a k du cas particulier ou $k = 2$ ..

Les corollaires 2 et 3 definissent un algorithme de fusion qui, pour $N = M$ et p quelconque est optimal a une constante pres si G est une clique; sinon, il reste optimal pour $2 \leq p \leq M + N$; ce qui represente la grande majorite des cas reels.

4. TRI DE N ELEMENTS.

Nous allons maintenant examiner l'implementation sur notre modele d'un tri de N elements par comparaisons partielles donne par Kruskal [KRU.] et Preparata [PRE.], et en deduire l'evolution des bornes de complexite.

4.1. L'algorithme, son implementation, et sa complexite dans le pire des cas:

L'algorithme de fusion du corollaire 3 permet de generaliser une idee de Preparata [PRE.] et de construire un algorithme de tri parallele rapide.

Ce tri, qu'on peut implementer sur une CREW-PRAM ou sur notre modele - dans tous les cas- est un tri dit par comparaisons partielles, ou suivant la terminologie de Knuth [KNU.] un tri par enumeration. Chaque element de la liste de taille N a trier est compare a tous les autres et le nombre d'elements inferieurs, determine le rang de l'element considere dans la liste hierarchique finale des elements tries.

Ce type de tri procede en plusieurs phases. Soit γ une constante (dependante de p et de N).

1. Si N = 1, la liste est triee;

2. Si p = 1, $Tri_1(N)$ trie en $Tri_1(N)$ pas de comparaison grace au meilleur algorithme sequentiel de tri.

3. Sinon on applique la procedure suivante :

a) *Numerotation des elements:* L'ensemble des elements est partitionne en sous-ensembles et, pour chaque element, on determine le nombre d'elements plus petit (on suppose tous les elements distincts);

b) *Calcul du rang:* Pour chaque element, on calcule la position finale de chaque element dans la liste triee;

c) *Rearrangement:* Chaque element est place selon son rang calcule, dans sa position finale.

En etant plus precis, l'algorithme suit en fait quatre phases;

i -L'ensemble des p processus est partitionne en γ groupes de $\frac{p}{\gamma} \geq 1$ processus chacun, et la liste des N elements est egalement partitionnee en γ groupes de $\frac{N}{\gamma} \geq 1$ elements chacun. (γ est de l'ordre de $\lfloor \log n \rfloor$.

ii - On trie recursivement chaque groupe independamment en parallele.

iii - On fusionne toutes les listes triees entre elles.

iv - On trie la liste en entier en considerant le rang de chaque element (cette phase demande $\binom{\gamma}{2}$ fusions independantes).

Cet algorithme necessite un nombre de comparaisons de l'ordre de

$$\frac{\log M}{\log \gamma} \cdot \left(\frac{N(\gamma-1)}{p} + \log \frac{N(\gamma-1)}{p} + loglogp \right)$$

(avec $M = \min \{p, N\}$),

et est optimal pour $\gamma = Max \{ \frac{3p \cdot loglog\ p}{ln3 \cdot N \log \gamma}, 2 \}$

On obtient ainsi des resultats qui ameliorent ceux de l'algorithme de Valiant et generalisent ceux de Preparata.

THEOREME : Si $N = p$, alors $\gamma = \frac{3}{ln3} \cdot \frac{loglog N}{logloglog N}$ et,

$$Tri_N(N) = 1{,}893 \cdot \frac{\log N \cdot loglog N}{logloglog N} \cdot (1 + O(\frac{logloglogloglog N}{logloglog N} + loglog\ N))$$

si G n'est pas une clique, et:

$$Tri_N(N) = 1{,}893 \cdot \frac{\log N \cdot loglog\ N}{logloglog\ N} \cdot (1 + O(\frac{loglogloglog\ N}{logloglog\ N}))$$

si G est une clique.

PREUVE:

Cette preuve, comme les suivantes consiste a faire $N = p$ dans la formule d'evaluation du nombre de comparaisons [KRU.].

(Cela represente une amelioration notable par rapport aux 2.logN.loglogN de Valiant).

Corollaire 1: Le choix optimal pour γ est 2; alors,

pour $N \geq \frac{3}{2 \cdot \ln 3} \cdot p \cdot loglog\ p$;

$Tri_p(N) \leq \frac{N \log N}{p} + \frac{3}{log 3} \log p \cdot loglog\ p + O(\frac{N}{p} + \log p \cdot \log\frac{2N}{p} + loglog\ p)$ si G n'est pas une clique, et:

$$Tri_p(N) \leq \frac{N \log N}{p} + \frac{3}{log 3} \log p \cdot loglog\ p + O(\frac{N}{p} + \log p \cdot \log\frac{2N}{p}$$

si G est une clique.

Corollaire 2: Si $p = \lceil N \cdot (log N)^{\frac{1}{k}} \rceil$; $(k \geq 2)$, alors:

$$Tri_p(N) \leq 1{,}893 \cdot klog N + O(klog N)\ ;$$ si G

est une clique, et:

$$Tri_p(N) \leq 1{,}893\ klog N + O(klog N + loglog\ p)$$

si G n'est pas une clique.

Ce dernier resultat, atteint sur notre modele, *ameliore* celui de Hirschberg [HIR.] montrant que $N^{1+\frac{1}{k}}$ processus peuvent trier en temps $O(klog N)$ et generalise celui de Preparata [PRE.]: NlogN processus peuvent trier en temps $O(log N)$.

Il est egalemant meilleur que celui de [SH,VI.] :

$Tri_p\ (N) = \bigcirc(k.logN.)$ si $p = \lceil N^{1+\frac{1}{k}} \rceil$ $(k > 1)$. Car du point de vue du nombre de processus p mis en oeuvre, on a

$\lceil N\ (logN.)^{\frac{1}{k}} \rceil$ processus ici, et

$\lceil N\ .\ N^{\frac{1}{k}} \rceil$ processus pour [SH,VI.] .

4.2. Remarques:

On trouvera l'algorithme detaille dans [PRE.].

Comme on peut le constater ici, aucun nouveau probleme d'implementation de cet algorithme ne se pose une fois resolus ceux afferants a l'algorithme de fusion.

Le modele ici propose, a la difference de la WRAM (et d'un anneau uni ou bidirectionnel), permet d'atteindre les meilleures bornes du tri de Kruskal, que G soit une clique ou non. De plus, par rapport a la CREW-PRAM, notre modele permet une implementation facile, que le graphe G des communications soit une clique ou un arbre.

BIBLIOGRAPHIE

[A,H,U.] AHO, HOPCROFT, ULLMAN "The Design and Analysis of Computer Algorithms" Addison-Wesley, Reading MASS, 1974.

[A,K,S] AJTAI, KOMLOS, SZEMEREDI "An O(nlogn) Sorting Network", 15-th Symp. on Th. of Comput. ACM, 1983.

[AT,SA,SA,ST.] ATKINSON, SACK, SANTORO, STROTHOTTE "An efficient, implicit double ended Priority Queue", SCS- TR.55 (July 84) Carleton Un. , School of Comput. Science. (Ottawa)

[BER] BERGE C. "Graphes et Hypergraphes", Dunod (1976).

[BLU] BLUM "A note on the 'Parallel Computation Thesis' ", Inf. Processing Letters 17. N.H pp.203-205 (1983).

[BO,HO.] BORODIN, HOPCROFT "Routing, merging and sorting on parallel models of computation"; 14th Symp. ACM. 1982. pp. 338-344.

[CAR] CARVALHO O. "Contribution a la Programmation Distribuee", These d'Etat, Inst. de Programmation - Univ. Paris VI (Paris 1985)

[CH,KO,ST.] CHANDRA A.K., KOZEN D.C., STOCKMEYER L.J., "Alternation", JACM Vol 27, (1981).

[C,S,V.] CHANDRA, STOCKMEYER, VISHKIN. "Complexity Theory of unbounded fan-in parallelism".Proc. 23 rd Annual IEEE Symp. on Found. of Comp. Science pp 1-13 1982

[FEL] FELLER "An Introduction to Probability Theory" , (vol.1) Wiley 1968.

[GA,JO.] GAREY, JOHNSON. "Computers and Intractability", Freeman (1979).

[GE,PU.] GELENBE, PUJOLLE. "Introduction aux Reseaux de files d'attente", Eyrolles. CNET-ENST (1982).

[G,G,K,M,R,S.] GOTTLIEB, GRISHMAN, KRUSKAL, Mac ANLIFFE, RUDOLF, SNIR. "The NYU Ultracomputer-designing a MIMD Shared memory Parallel Machine" IEEE Trans. on Comput., C-32, 2, 1983, pp 175-189.

[JOH.] JOHNSON "The NP-completeness Colomn: an Ongoing guide" Journal of Algorithms 5, 1984 (pp.595-609)

[KNU.] KNUTH D. E. "The art of computer programming" Vol. 1 & 3 "Sorting and searching" Addison Wesley Publishing Company.

[KOZ.] KOZEN D., "On Parallelism in Turing machines" - 17th IEEE symposium on foundation of Computer Science - 1976.

[KRU] KRUSKAL "Results in Parallel Searching, Merging and Sorting" (Summary), Proc. Int. Conf. on parallel Processing - 1982

[KR,WE.] KRUSKAL, WEISS "Allocating independant subtasks on Parallel Processors", Proc. 13-rd Conf. on Parallel Processing. IEEE (1984).

[LA,LA.] LAVALLEE I., LAVAULT C. "Scheme for Complexity Measures of Distributed and Parallel Algorithms" Summer School on Combinatorics Federal University of Rio de Janeiro, July 1985.

[LA,RO.1] LAVALLEE I., ROUCAIROL G."A Fully Distributed (minimal) Spanning Tree Algorithm" Internal Report, L.R.I. Univ. Paris-Sud, Bat. 490 - 91405 ORSAY Cedex/FRANCE.

[LA,RO.2] LAVALLEE I., ROUCAIROL Cath."A Parallel Branch and Bound Algorithm" EURO VII congress, Bologne Juin 1985.

[LO,LA.] LOUCHARD, LATOUCHE "Probability Theory and Computer Science", Academic Press 1983.

[PA,KO,RO.] PACHL, KORACH, ROTEM "Lower Bounds for Distributed Maximum Finding Algorithms" JACM, Vol. 31, No 4-1984, pp. 905 918.

[PA,PR,RE.] PAUL W.J., PRAUSSE J., REISCHUK R., "On Alternation" Acta Informatica 14, 243-255 (1980).

[PA,RE.] PAUL W.J., REISCHUK R., "On Alternation II" Acta Informatica 14,

391-403, (1980).

[PRE.] PREPARATA "New Parallel Sorting Schemes", IEEE Trans. on Comp., Vol. C-27, No 7, 1978.

[RAB.] RABIN "Probabilistic Algorithms" Alg. and complexity, J.F. Traub editor, Academic press 1976 pp 21-39.

[RE,VA.] REIF, VALIANT "A Logarithmic time sort for linear size networks" 15th ACM Symp. on Found. of Comput. -1983.

[RO,SA,SI.] ROTEM D., SANTORO N., SIDNEY J.B. "Distributed sorting" IEEE Trans. on Comp. Vol. C-34, No 4, (April 1985).

[SAN.] SANTORO "On the message complexity of distributed problems" Distributed Computing Group Carleton University Ottawa - Dec.1982.

[SH,VI.] SHILOACH, VISHKIN. "Finding the Maximum, Merging and Sorting in a Parallel Computation Model"; Journal of Algorithms 2, 1, Mach 81, pp 88-102.

[STO.] STOUT "Sorting, Merging, Selecting and Filtering on tree and Pyramid Machines" (preliminary version) Proceeding conference on Parallel Processing IEEE -1983.

[TH,BA.] THOMASSIAN, BAY "Queuing networks models for parallel processing of task system" Proc. Conf. on Parallel Processing 1983 - pp 421, 428.

[UPF.] UPFAL "A Probabilistic relation between desirable and feasible models of parallel computation" (preliminary version) 16th ACM symp. on the Found. of Comp. 1984.

[VAL.] VALIANT "Parallelism in Comparison Problems"; SIAM J. Comp., 4, 1975, pp. 21-39.

[VIS.1] VISHKIN "Synchronous parallel computation - A survey" Techn. Rep. 71 Dept. of Comp. Sc. Courant Institute, NY University (April 1983).

[VIS.2] VISHKIN "Randomized speed-ups in parallel computation" 16th Proc. of ACM on theory of computing 1984.

[VI,WI.] VISHKIN, WIDGERSON "Trade-offs between depth and width in Parallel Computation" (Preliminary Version), 24th Symp. on Found. of Comput. Science IEEE, 1983 (pp. 146-153).

[WA,MA.] WAH B.W. and MA Eva Y.W., "MANIP - A Multicomputer Architecture for Solving Combinatorial Extremum-Search Problems", IEEE Trans. on Comp., Vol. c-33, No. 5, May 1984, pp 377-390.

[YAO] YAO "Lower bounds by probabilistics arguments" 24th Symp. IEEE on foundation of Computer Science 1983, pp 420, 428.

ANNEXE

EQUIVALENCE ENTRE loglog (p) ET $\log_m$ (p)

Dans la suite, p represente le nombre de processus et m est le degre de l'arbre *parfait* sous-jacent a' la structure "en grappe" : on a la relation

$$m = \lceil e^{(\frac{\ln p}{loglog\ p})} \rceil$$

("ln" signifie logarithme neperien, et "log" logarithme en base 2)

p=	loglog p=	m=
10	1,7	4
50	2,5	5
10^2	2,7	6
10^3	3,3	9
10^4	3,7	12
10^5	4	18
10^6	4,3	25
10^9	4,9	69
10^{10}	5	100
10^{20}	6	2155
10^{99}	8,36	$6{,}9.\ 10^{11}$

Le tableau ci-dessous montre bien que, pour 100 000 processus par exemple, il suffit de construire une structure avec un 18-arbre parfait sous-jacent pour atteindre la hauteur 4, et ... $loglog(10^5)$. Dans une telle structure, les communications entre processus sont en $\mathcal{O}(loglog(p))$.

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