RAIRO. RECHERCHE OPÉRATIONNELLE

J. P. BOUFFLET
J. CARLIER

Décomposition d'une application sur une architecture bus : propriétés des ordonnancements optimaux

RAIRO. Recherche opérationnelle, tome 31, n° 1 (1997), p. 17-43
<http://www.numdam.org/item?id=RO_1997__31_1_17_0>

Recherche opérationnelle/Operations Research
(vol. 31, n° 1, 1997, pp. 17-43)

DÉCOMPOSITION D'UNE APPLICATION SUR UNE ARCHITECTURE BUS : PROPRIÉTÉS DES ORDONNANCEMENTS OPTIMAUX (*)

par J. P. Boufflet ([1]) et J. Carlier ([1])

Résumé. – *Cet article porte sur des problèmes liés à l'exécution d'une application sur une machine multiprocesseurs à architecture bus ayant un processeur maître et des processeurs esclaves. Le problème le plus sérieux est la congestion du bus : à tout instant seul un des processeurs esclaves peut communiquer avec le maître. Un second problème est de maintenir actifs les processeurs. L'application est supposée décomposable en tâches qui seront envoyées aux processeurs esclaves par le maître. De plus, par hypothèse, le temps de calcul de chaque tâche est une fonction linéaire de son volume de données. L'objectif est de minimiser la durée totale de l'exécution en tenant compte des temps de calcul et des temps de communication. Nous étudions d'abord les propriétés des ordonnancements optimaux. Puis nous nous intéressons à deux cas particuliers avec des temps de calcul proportionnels aux volumes de données : un processeur recevant plusieurs tâches et deux processeurs recevant une seule tâche. Pour conclure, nous remarquons que la méthodologie que nous avons élaborée pourrait s'appliquer sous des hypothèses plus générales.*

Mots clés : Multiprocesseur, bus, maître, esclaves, décomposition en tâches, ordonnancement.

Abstract. – *A bus oriented multiprocessor contains one bus system to which all the processors are connected. There are one master and several slaves. The most serious problem is the bus bottleneck. At a time, only one slave can establish communications through the bus with the master. A second problem is to maintain busy each processor. We suppose we have to execute a task which uses a big amount of data and which can be split into several independent processes. Moreover, the processing time of each process is a linear function of its amount of data. The objective is to minimize the makespan by taking into account communication durations and processing times. At first, the properties of optimal schedules are studied. Next, two particular cases with task computation times proportional to their amounts of data are considered. In the first case, we have one slave and several processes sent to it. So, the procèssor can compute in masked time after the reception of the first amount of data. In the second case, we have two slaves and only one process is sent to both of them. Finally, we remark that this approach can be generalized.*

Keywords: Multiprocessor, bus, master, slaves, task decomposition, scheduling.

(*) Ce travail a été financé par la région Picardie dans le cadre du projet parallélisme du pôle régional de modélisation. Il s'effectue dans le cadre du réseau Capital Humain et Mobilité « Models, Algorithms and Systems for Decision Making », contrat numéro ERB-CHRX-CT93-0087, mis en œuvre par la Communauté Économique Européenne. Reçu en décembre 1993.

([1]) URA 817 HEUDIASYC U.T.C. Génie Informatique, 60205 Compiègne Cedex.
E-mail: carlier@hds.univ-compiegne.fr or boufflet@hds.univ-compiegne.fr

Recherche opérationnelle/Operations Research, 0399-0559/97/01/$ 7.00

1. INTRODUCTION

Bien que la puissance des ordinateurs croissent régulièrement, il est patent qu'une limite physique sera un jour atteinte. Pour remédier à ce problème, et sous l'effet de la demande sans cesse croissante de puissance de calcul, on a cherché à inventer d'autres architectures d'ordinateurs que la classique architecture de Von Neumann. Le point commun de ces architectures est l'utilisation de plusieurs processeurs. Mais on constate empiriquement que l'utilisation moyenne de la puissance de calcul est bien inférieure à la puissance de crête de ces nouvelles machines. Une des raisons est le problème des temps de communication entre les processeurs ([COT 93], [BET 89], [CHR 92]).

Chaque machine parallèle dispose au sein de son logiciel système d'un algorithme d'allocation de tâches et de gestion des communications. Actuellement, les algorithmes d'allocations ne sont pas adaptés à tous les types de traitements. Cela tient évidemment au non déterminisme des programmes, mais aussi au fait que l'on ne connaît pas les lois de décomposition d'une application en tâches, permettant d'équilibrer la charge des processeurs et de minimiser les communications ([FRA 92], [BCS 91], [MUN 91]). Que le programmeur découpe lui-même l'application en tâches, ou qu'il laisse le logiciel système décider, il est donc nécessaire de dégager des règles générales de planification des tâches [CAC 88]. Le but de cet article est de présenter un cas simple qui servira de base de réflexion pour dégager de telles règles.

Quand on exécute une application sur une architecture parallèle à N processeurs, on espère diviser le temps de calcul de cette application par le nombre N de processeurs. Cela n'est pas possible pour plusieurs raisons. Il y a d'abord des problèmes système. Toutefois ces problèmes système ne sont pas insurmontables quand on a un ordinateur hôte qui gère le système, et des processeurs esclaves qui exécutent l'application. Nous nous plaçons sous cette hypothèse par ailleurs très réaliste en nous intéressant au cas d'une architecture bus [TAB 90] qui est la forme la plus communément répandue de parallélisme. En effet, un réseau standard de stations a une telle structure.

Il faut aussi considérer les problèmes liés au non déterminisme des programmes. On ne peut pas toujours connaître le temps-calcul de chaque partie de l'application. S'il en est ainsi, il faut gérer « on-line », c'est-à-dire par des méthodes dynamiques. Nous nous placerons sous l'hypothèse que l'application est parfaitement connue et a un comportement déterministe. Il en est ainsi pour de nombreuses applications d'analyse numérique [ESC 91].

Il y a ensuite les contraintes logiques reliant les différentes tâches de l'application. En effet, une application peut être considérée comme un ensemble de tâches communiquant entre elles et liées par des relations de précédence qui sont dues à l'utilisation de variables communes. Ces contraintes limitent de façon évidente le parallélisme. Nous ne tiendrons pas compte de ces contraintes de synchronisation entre les tâches. Cette hypothèse est réaliste dans la mesure où une classe de problèmes lui correspond.

En effet, on peut avoir à faire des calculs identiques sur un grand nombre d'objets, par exemple faire de la compression d'images. Dans ce cas on affectera aux processeurs un sous-ensemble d'images et on rapatriera les images compressées. On peut également vouloir tester une propriété d'un grand nombre d'objets indépendants, par exemple, la planarité de graphes ou la primarité de nombres entiers. Un troisième exemple est la méthode multifrontale des éléments finis où on introduit autant de sous-domaines que de processeurs, pour traiter indépendamment chaque sous-domaine par des opérations matricielles, puis on rapatrie les résultats intermédiaires pour finalement résoudre le système linéaire global.

En résumé, on suppose que l'on a une application s'exécutant sur une machine formée d'un maître et de N processeurs liés par un bus dont la structure peut se représenter symboliquement comme sur la figure 1.

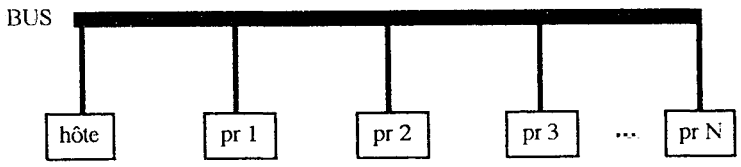

Figure 1. – L'architecture bus.

Les paramètres de l'architecture bus sont les suivants :

- N est le nombre de processeurs disponibles ;
- la mémoire locale de chaque processeur est suffisante ;
- les processeurs ne gèrent pas leurs communications, elles sont gérées par un système dédié local à chaque processeur ;
- les systèmes dédiés gérant les communications ont une mémoire tampon de taille suffisante qui leur permet de stocker plusieurs travaux à la fois ;
- d_j est le temps d'établissement de la communication du maître (resp. de l'esclave j) vers l'esclave j (resp. le maître), tous les d_j sont strictement positifs ;

- $\frac{1}{\alpha}$ est la vitesse du bus en transmission (sens maître vers esclaves) ;
- $\frac{1}{\beta}$ est la vitesse du bus en rapatriement (sens esclaves vers maître) ([2]).

En outre nous faisons les hypothèses suivantes en ce qui concerne l'application :

- il n'y a pas de contraintes de synchronisation du moins jusqu'à un certain grain raisonnable des tâches ;
- l'application utilise un volume connu de données et est décomposable en un nombre aussi grand que l'on veut de tâches indépendantes ;
- chaque tâche sera affectée à un processeur et sa durée de calcul est une fonction linéaire de son volume de données dont les paramètres dépendent de ce processeur et de l'application.

Le but de l'étude est de réaliser le découpage de l'application et son ordonnancement sur cette architecture en minimisant le temps total $C_{\max}$ d'exécution. Cela nécessite de déterminer le nombre optimal de processeurs à faire travailler ainsi que la meilleure quantité de données à envoyer à chaque processeur. Pour réussir cette optimisation, il faut tenir compte de paramètres directement liés à la machine-bus, c'est-à-dire les temps d'établissement des communications (set-up times) de chaque processeur au processeur maître, la bande passante du bus et son nombre de processeurs. Notre objectif est d'étudier les propriétés que satisfont certaines solutions optimales et de caractériser dans des cas particuliers ces solutions optimales.

Au paragraphe 2, nous récapitulons le problème. Au paragraphe 3, nous montrons des propriétés préliminaires des ordonnancements optimaux qui nous amènent à introduire des notations au paragraphe 4. Ceci nous permet de démontrer des propriétés complémentaires au paragraphe 5. Au paragraphe 6, nous étudions le cas particulier de plusieurs envois à un seul processeur. Puis le cas de deux processeurs, avec un seul envoi par processeur, est détaillé au paragraphe 7. Nous discutons des extensions de cette approche dans les cas où les temps de calcul ne sont plus des fonctions linéaires au paragraphe 8. Enfin, nous concluons et présentons les perspectives de ce travail au paragraphe 9.

2. MODÈLE ÉTUDIÉ

Le hardware est constitué de N processeurs esclaves $1, 2, \ldots, N$ liés par

([2]) En fait, la vitesse du bus en transmission et pour le rapatriement est la même. Mais le fait d'avoir $\beta = \alpha\mu$ avec $\mu << 1$ a pour but de traduire le fait que l'on rapatrie moins de données que l'on en envoie. Ce qui est par exemple le cas dans l'exemple de la compression des images.

un bus à un processeur maître, tel qu'il a été présenté dans la figure 1 de l'introduction. Les temps d'établissement des communications (set-up times) du maître vers les esclaves et des esclaves vers le maître sont respectivement $d_1, d_2, \ldots, d_N$. Ces quantités seront appelées abusivement distances par la suite. Chaque processeur a une mémoire locale suffisante.

On doit exécuter une application sur la machine correspondante. Cette application utilise un volume de données P connu et peut être divisée en plusieurs tâches exécutables en parallèle et travaillant sur des données disjointes. Le volume P de données est continûment décomposable en autant de morceaux que nécessaire.

Ces tâches sont envoyées aux divers processeurs, et on suppose que :

- sur les N processeurs de la machine, n seulement sont actifs ;
- la durée de l'envoi i, de volume p_i, au processeur j est $x_i = \alpha p_i + d_j$;
- l'envoi i, qui a un volume p_i, est stocké dans un buffer ;
- il est possible que plusieurs données soient envoyées successivement à un même processeur ;
- pendant tous les envois, y compris ceux qui lui sont destinés, un processeur peut travailler ;
- pour travailler sur l'envoi i, il est nécessaire que celui-ci soit entièrement arrivé ;
- le calcul à effectuer a une durée $g_j(p_i)$ où g_j est une fonction linéaire à coefficients positifs dépendant du processeur j sur lequel la tâche correspondante s'exécute et de la nature de l'application. Donc $g_j(p_i) = \lambda_j p_i + \omega_j$ où λ_j dépend du type de l'application (qui doit être de complexité linéaire) et de la vitesse d'exécution du processeur. Enfin, ω_j représente le temps constant d'établissement du contexte de travail sur le processeur ;
- le rapatriement du travail effectué par le processeur j est global, donc dans le cas où le processeur j reçoit plusieurs envois de données, il faut attendre que le dernier travail demandé au processeur soit terminé avant de rapatrier l'ensemble des résultats ;
- la durée du rapatriement numéro h d'un volume p de données du processeur j est $y_h = \beta p + d_j$.

Il faut choisir la décomposition de l'application en tâches et l'ordonnancement de ces tâches sur la machine-bus.

3. PROPRIÉTÉS PRÉLIMINAIRES

Ces propriétés sont dues au fait qu'il n'y a de limitation ni sur les mémoires locales des processeurs, ni sur les mémoires des systèmes dédiés aux communications. On recherche une solution qui minimise le temps d'occupation du bus. On peut supposer sans perte de généralité que :

PROPRIÉTÉ 1 : *L'ensemble des envois précèdent l'ensemble des rapatriements.*

Démonstration : Raisonnons par l'absurde. Sinon un rapatriement précède immédiatement un envoi. Les deux occupations correspondantes du bus peuvent être éventuellement séparées par une période d'inoccupation de ce même bus. Transposons l'envoi et le rapatriement et étudions les répercussions de cette opération (*fig.* 2).

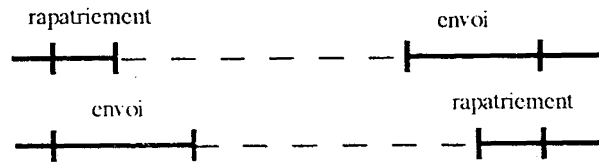

Figure 2. – Les envois précèdent les rapatriements.

Cette opération est licite car on peut, d'après les hypothèses, stocker les données des tâches et les rapatriements à effectuer dans le tampon du système dédié aux communications de chaque processeur.

Les contraintes sont respectées sans modification de la durée totale de l'ordonnancement. En effet, l'envoi est disponible plus tôt pour le processeur qui l'exécute et le rapatriement a lieu plus tard. On peut réappliquer cette procédure jusqu'à obtenir un ordonnancement dans lequel tous les envois précèdent tous les rapatriements.

Q.E.D.

PROPRIÉTÉ 2 : *Les envois sont au plus tôt et les rapatriements au plus tard.*

Cette propriété se démontre en avançant les envois et en retardant les rapatriements, on a toujours la possibilité d'effectuer les envois au plus tôt et les rapatriements au plus tard.

La prise en compte de ces deux propriétés nous conduit à ne considérer que les ordonnancements suivants : on a d'abord tous les envois sans

temps intercalaire d'inoccupation du bus, puis éventuellement un temps d'inoccupation du bus, et enfin l'ensemble des rapatriements de données.

4. NOTATIONS COMPLÉMENTAIRES

Les deux propriétés préliminaires permettent de visualiser l'utilisation du bus (*fig.* 3).

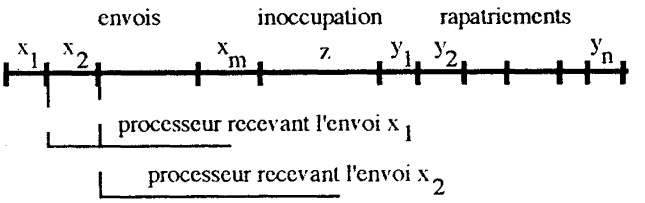

Figure 3. – Diagramme de Gantt d'un ordonnancement.

En conséquence, dans un ordonnancement optimal, le bus sera initialement occupé par les m envois de données aux processeurs de durées $x_1, x_2, \ldots, x_m$, puis il y aura éventuellement une période z d'inoccupation du bus, et enfin les rapatriements des résultats des n processeurs de durées $y_1, y_2, \ldots, y_n$.

Donc la solution d'un tel problème est caractérisée par la recherche :

- d'un découpage de l'application en un nombre m de tâches à déterminer ;
- de la taille des m paquets p_i de données à envoyer aux processeurs ;
- du nombre n de processeurs à faire travailler ;
- de l'affectation des envois aux processeurs : $a(i)$ est le numéro du processeur recevant l'envoi i ;
- de l'ordre des rapatriements : $b(j)$ est le numéro de rapatriement du processeur j.

On a compte tenu des hypothèses les équations suivantes :

- la quantité totale de données de l'application est égale à la somme des paquets de données envoyées : $P = \sum_{i=1}^{m} p_i$;
- la durée d'occupation du bus pour l'envoi i est donnée par : $x_i = \alpha\, p_i + d_{a(i)}$;
- la durée du calcul correspondant à l'envoi i au processeur j est : $g_j(p_i) = \lambda_j p_i + \omega_j$;
- la durée du rapatriement du processeur j est : $y_{b(j)} = \beta \sum_{a(k)=j} p_k + d_j$, ce qui correspond au rapatriement numéro h, avec $h = b(j)$;

• la durée totale de l'ordonnancement est donc :

$$C_{\max} = \sum_{i=1}^{m} x_i + z + \sum_{k=1}^{n} y_k.$$

$C_{\max}$ est la quantité que nous cherchons à minimiser.

5. PROPRIÉTÉS COMPLÉMENTAIRES

Afin d'établir des propriétés complémentaires, nous introduisons la définition de la signature d'un ordonnancement.

DÉFINITION 1 : On appelle signature d'un ordonnancement le quadruplet $(n,\ m,\ a,\ b)$ où n est le nombre de processeurs actifs, m le nombre d'envois, a l'application $[1, m] \to [1, n]$ fixant l'affectation des envois aux processeurs et b la permutation sur $[1, n]$ correspondant à l'ordre des rapatriements.

PROPRIÉTÉ 3 : *Il existe une solution de durée minimale.*

Pour démontrer cette propriété, nous énonçons et prouvons le lemme suivant :

LEMME : *Pour une signature donnée on a une solution optimale.*

Démonstration du lemme : On vérifie d'abord qu'à toute décomposition $p_1, p_2, \ldots, p_m$ de P avec $p_1 \geq 0, p_2 \geq 0, \ldots, p_m \geq 0$ et $\sum_{i=1}^{m} p_i = P$ correspond une solution de $C_{\max}$ minimale associée à un problème d'ordonnancement à contraintes potentielles [CAC 88]. En effet, du fait que l'on se limite à une signature, on a à exécuter $2m + n$ tâches, dont les durées sont fixées et qui ne sont liées que par des contraintes de précédence. Il y a m tâches d'envoi, m tâches de calcul et n tâches de rapatriement. Les m tâches d'envoi sont séquencées dans l'ordre $1, 2, \ldots, m$. Les tâches de calcul envoyées à un processeur sont séquencées dans leur ordre d'envois. Les tâches de rapatriement sont séquencées par la fonction b. En conséquence les ordonnancements au plus tôt et au plus tard sont définis. C'est sur l'ordonnancement au plus tôt que l'on raisonne pour les tâches d'envoi et de calcul, et sur l'ordonnancement au plus tard pour les tâches de rapatriement. Il est possible de raisonner ainsi car les tâches de rapatriement suivent toutes les autres tâches. $C_{\max}$ dépend continûment de $p_1, p_2, \ldots, p_m$ sur le polyèdre définissant les p_i. On a donc pour une signature donnée une solution optimale.

Démonstration de la propriété 3 : On montre facilement, puisque les d_j sont strictement positifs, que le nombre de signatures à considérer est fini. En effet, si on a une solution de durée $C_{\max}$ et si on note $d = \min\{d_j/j \in [1, N]\}$, toute signature ayant plus de $C_{\max}/d$ envois n'aura pas à être considérée car alors le bus est déjà occupé pendant une durée supérieure à $C_{\max}$. Il en résulte évidemment l'existence d'une solution optimale puisqu'il y a un nombre fini de cas à considérer.

Q.E.D.

Avant d'introduire et de démontrer la propriété 4, il nous est nécessaire d'introduire la définition d'un envoi trop important. Nous considérons pour cela deux envois successifs e et f à un même processeur j et nous notons C_e et C_f leurs dates de fin.

DÉFINITION 2 : Un envoi est dit trop important pour un processeur j si la quantité transmise à ce processeur j est plus grande que la quantité calculable par ce même processeur jusqu'à réception de l'envoi suivant, c'est-à-dire si : $C_f - C_e < g_j(p_e)$.

Ceci est représenté symboliquement sur la figure 4 qui n'est pas un véritable diagramme de Gantt, dans la mesure où ne sont pas représentés les envois antérieurs à e, qui peuvent retarder son exécution.

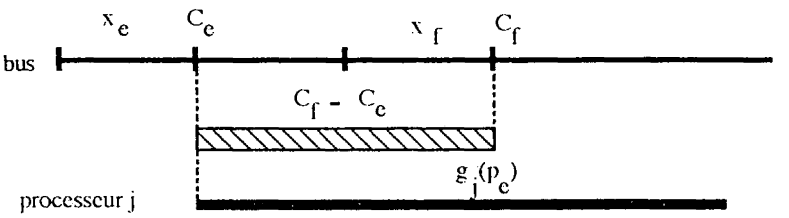

Figure 4. – Un envoi trop important.

PROPRIÉTÉ 4 : *Il existe un ordonnancement optimal tel qu'aucun envoi est trop important.*

Démonstration : Considérons un ordonnancement optimal ayant un nombre minimal d'envois et observons un processeur j ayant un envoi trop important. Pour pallier à un envoi trop important nous définissons ci-dessous une opération qui consiste à diminuer l'envoi trop important d'une quantité que l'on reporte sur le prochain envoi à ce même processeur j. La quantité calculée par le processeur j reste la même, seules changent la date de fin du premier envoi, et la date de début du second envoi. Nous allons utiliser cette opération de base pour démontrer la propriété 4.

Considérons e et f deux envois à un même processeur j de dates de fin C_e et C_f, tels que le premier envoi est trop important, c'est-à-dire $C_f - C_e < g_j(p_e)$. Remarquons d'abord que l'on peut supposer que $C_f - C_e \geq g_j(0) = \omega_j$ car la solution est optimale. En effet, s'il n'en était pas ainsi, l'envoi x_e serait inutile et on pourrait le supprimer. Cette condition s'écrit $0 \geq g_j(0) - g_j(p_e) + D_e$, en définissant D_e par : $D_e = g_j(p_e) - C_f + C_e$. Nous allons utiliser l'opération de base pour faire disparaître l'écart D_e (*fig.* 5).

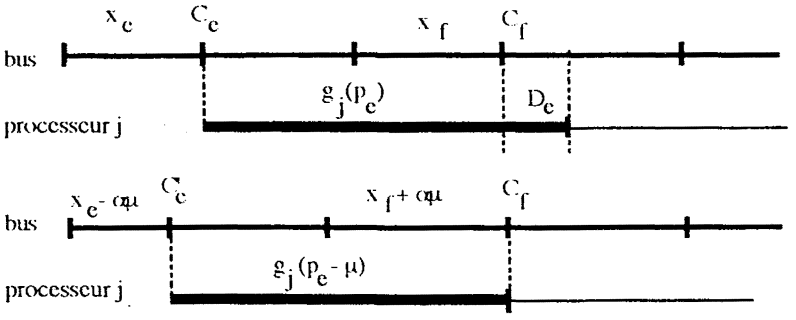

Figure 5.

Dans cette transformation, les envois compris entre les envois e et f sont avancés. De plus les envois antérieurs à e et postérieurs à f n'ont pas leur fonctionnement perturbé si ce n'est que l'envoi au processeur j antérieur à x_e peut devenir trop important. On garde donc l'admissibilité et on peut même diminuer $C_{\max}$ car certaines opérations avancent.

On va montrer que l'on peut envoyer une quantité $p'_e = p_e - \mu$, qui vérifie $C_f - C'_e = g_j(p_e - \mu)$, c'est-à-dire que l'on n'envoie pas trop. Pour cela nous introduisons ci-dessous une fonction $h_j(\mu)$ et montrons que cette fonction s'annule. Le résultat en découlera. $h_j(\mu)$ est par définition le dépassement en valeur relative de la date C_f par la fin de l'exécution de l'envoi x'_e sur le processeur j. C'est-à-dire, $h_j(\mu) = C'_e + g_j(p_e - \mu) - C_f$, donc : $h_j(\mu) = C_e - \alpha\mu + g_j(p_e - \mu) - (C_e + g_j(p_e) - D_e)$, d'où : $h_j(\mu) = g_j(p_e - \mu) - g_j(p_e) + D_e - \alpha\mu$.

Nous remarquons que $h_j(0) = D_e$. Montrons que $h_j(p_e) < 0$: en effet, $h_j(p_e) = g_j(0) + D_e - \alpha p_e - g_j(p_e)$ qui est strictement négative car $0 \geq g_j(0) - g_j(p_e) + D_e$ et $-\alpha p_e < 0$. La fonction h est continue et change de signe sur le segment $[0, p_e]$, il existe donc un μ tel que $h_j(\mu) = 0$.

Alors la date de fin d'exécution de l'envoi x'_e coïncide avec la date d'arrivée de l'envoi x'_f.

Considérons dans un ordonnancement optimal ne vérifiant pas la propriété 4, les couples d'envois e et f pour lesquels D_e est maximum, c'est-à-dire le temps de chevauchement de deux envois est maximal, avec en cas d'ex-aequos C_e minimale. Quand on applique la transformation définie ci-dessus, on obtient un nouvel ordonnancement optimal. En itérant, on construit une suite d'ordonnancements optimaux.

On considère la suite $\sum_{i=1}^{m} C_i$ associée à cette suite d'ordonnancements. Cette suite est évidemment strictement décroissante. Il en résulte qu'elle converge car elle est minorée par 0. Donc au bout d'un certain temps, on la rend aussi proche que l'on veut de sa valeur limite. Chaque C_i converge également, on obtient donc un ordonnancement limite qui, nous allons le voir, vérifie la propriété 4.

D'après ce qui précède, on a pour tout ordonnancement de la suite : $D_e = g_j(p_e) - g_j(p_e - \mu) + \alpha\mu$. Donc D_e est une fonction continue de μ qui prend la valeur 0 pour $\mu = 0$. Comme on s'est placé dans le cas linéaire, D_e ne dépend pas de p_e mais de j et de μ, donc quand μ tend vers 0, D_e tend vers 0. En conséquence, quand $\sum_{i=1}^{m} C_i$ tend vers sa valeur limite, μ tend vers 0 et D_e tend vers 0.

Q.E.D.

Avant de démontrer la cinquième propriété, nous énonçons deux nouvelles définitions.

DÉFINITION 3 : Un volume de données envoyé à un processeur j est dit insuffisant s'il ne lui permet pas de travailler jusqu'à la réception du prochain volume de données qui lui est destiné (*fig.* 6). Dans ce cas on a $C_f - C_e > g_j(p_e)$.

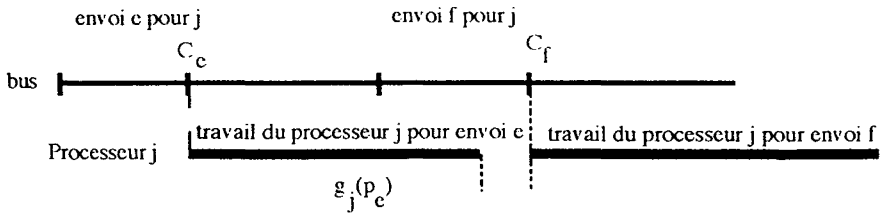

Figure 6. – L'envoi e est insuffisant.

DÉFINITION 4 : Un volume de données est dit utile si on a $C_f - C_e = g_j(p_e)$.

PROPRIÉTÉ 5 : *Dans une solution optimale pour laquelle aucun envoi n'est trop important si le bus n'est pas actif durant toute la durée de l'ordonnancement (bus non critique), le volume de données de chaque envoi est exactement égal à celui utile, c'est-à-dire ni insuffisant ni trop important.*

Démonstration de la propriété 5 : On démontre la propriété 5 par des arguments liés à la programmation linéaire. Nous allons voir que la minimisation de la durée totale $C_{\max}$ se ramène à la résolution d'un programme linéaire ayant $2m+3$ inéquations principales et $m+2$ variables principales.

On distingue ci-dessous les variables principales des variables secondaires, les conditions de positivité, les inéquations associées aux envois, l'équation de conservation du volume de données et l'équation associée au bus.

Les $p_i, C_{\max}$ et z sont choisis comme variables principales ($m+2$ variables). En effet les autres variables (x_i et y_j) peuvent s'exprimer en fonction de celles-ci par les formules vues précédemment (*cf.* paragraphe 4). Les quantités envoyées sont positives : $p_i \geq 0$ (i variant de 1 à m). Ces m inéquations sont strictement positives dans la solution optimale, car on n'a aucun intérêt à faire un envoi « vide ». Il y a une équation de conservation : $P = p_1 + p_2 + \ldots + p_m$.

Les équations associées au bus sont : $z = C_{\max} - x_1 - \ldots - x_m - y_1 - \ldots y_n$ et $z \geq 0$. On doit associer à chaque envoi l'inéquation suivante : $\sum_{l=1}^{i} x_l + \sum_{a(i')=a(i) \text{ et } i' \geq i} p_{i'} + \sum_{h \geq k \text{ et } b(a(i))=k} y_h \leq C_{\max}$. La somme des x_l représente l'ensemble des envois antérieurs à x_i (i inclus). La somme sur les $p_{i'}$ représente l'ensemble des calculs effectués sur le processeur j sur les envois postérieurs à x_i (i inclus). La dernière somme donne la durée des rapatriements des résultats des processeurs ayant leur numéro de rapatriement postérieur au rapatriement du processeur j. Nous avons donc m inéquations supplémentaires, donc un programme linéaire de $2m+3$ inéquations ou équations, de $m+2$ variables principales, la fonction à minimiser est la durée $C_{\max}$ d'occupation du bus.

Dans un programme linéaire à l'optimum, on a au moins autant d'inégalités serrées (c'est-à-dire d'inégalités qui sont des égalités) que de variables du programme linéaire. Les inégalités de positivité des variables p_i sont nécessairement strictes et nous considérons le cas $z > 0$ (bus non critique). Il reste $m+2$ inégalités ou égalités et $m+2$ variables, donc à l'optimum les inégalités associées aux envois sont des égalités. Ce qui permet de conclure que le volume de chaque envoi est juste égal à celui utile (*cf.* définition 4).

Q.E.D.

Nous utiliserons ce résultat dans les paragraphes 6 et 7 de cet article. La pro- priété 5 est typiquement une règle générale de planification des tâches. Cette règle consiste à dire qu'il faut envoyer à chaque processeur exactement la quantité de travail qui lui est nécessaire pour qu'il puisse travailler jusqu'au prochain envoi.

Pour le cas d'un bus critique, on a cette fois $z = 0$ et une inéquation peut être stricte, d'où le corollaire suivant :

COROLLAIRE : *Dans une solution optimale pour laquelle aucun envoi n'est trop important, si le bus est actif durant toute la durée de l'ordonnancement (bus critique), il y a une solution optimale pour laquelle le volume de données de chaque envoi est exactement égal à celui utile à l'exception du volume d'au plus un envoi qui peut être insuffisant.*

DÉFINITION 5 : On appelle contamination à gauche le fait de transférer une fraction de l'envoi i-1 à l'envoi i. Sous l'hypothèse d'indépendance logique entre les données, on peut faire traiter indifféremment des données par des processeurs différents. La figure 7 montre les états du bus avant et après la contamination à gauche.

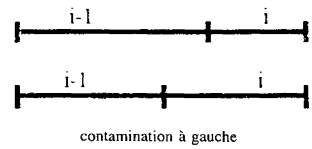

Figure 7. – Contamination à gauche.

Plaçons-nous dans le cas où $\beta = 0$, donc le temps du rapatriement ne dépend que de la distance au processeur maître. Cela correspond concrètement au cas où l'on cherche à vérifier une propriété sur un grand ensemble d'objets. Il y a beaucoup de données à envoyer et les réponses obtenues par l'application sont de types booléennes, on peut considérer que $\beta = 0$, car on ne rapatrie pratiquement pas de données.

PROPRIÉTÉ 6 : *Dans le cas $\beta = 0$, on peut supposer que c'est le premier envoi qui est insuffisant.*

Démonstration de la propriété 6 : Pour démontrer la propriété 6, la technique consiste à contaminer les voisins de gauche de façon à ce que seul le premier envoi soit insuffisant, l'envoi qui contamine devient alors juste suffisant.

Q.E.D.

L'ensemble des propriétés que nous venons d'exposer définissent un cadre général de travail. A partir de ce cadre nous pouvons explorer différentes voies. En effet, il faut maintenant proposer des méthodes de résolution. Par exemple, on peut essayer de construire une méthode exacte d'ordonnancement des tâches sur les processeurs. Pour cela on est amené à émettre des hypothèses sur les fonctions $g_j(p_i)$. On peut également élaborer des heuristiques et les évaluer.

Nous allons maintenant étudier, dans les paragraphes 6 et 7, deux cas particuliers qui respectent l'ensemble des propriétés. Une hypothèse supplémentaire est faite sur les fonctions $g_j(p_i)$. Elles sont simplement de la forme $g_j(p_i) = p_i$.

6. UN PROCESSEUR RECEVANT PLUSIEURS ENVOIS

Considérons le problème relatif à un processeur à distance d du maître, recevant plusieurs envois. Nous allons étudier successivement les cas de deux envois, trois envois, ..., m envois. Il faut, pour un volume de données P fixé, choisir le nombre d'envois qui minimise $C_{\max}$. Il s'agit de pipeliner le volume de données et donc les messages de telle sorte que la durée totale du calcul soit minimale. Nous allons établir les formules qui régissent ce cas particulier. Remarquons d'abord qu'il y a nécessairement une période d'inoccupation du bus. En effet, celle-ci est en correspondance avec le dernier envoi (*fig.* 8). Par ailleurs, la durée du rapatriement est égale à $\beta P + d$ et donc indépendante de la décomposition. On prendra $\beta = 0$ sans perte de généralité. En conséquence, on peut appliquer la propriété 5, et calculer pour une signature donnée une répartition optimale en résolvant un système linéaire.

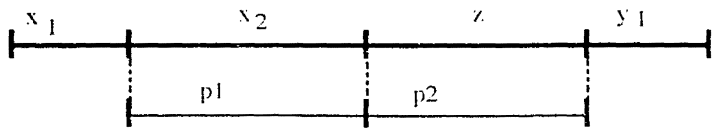

Figure 8. – Cas linéaire, un processeur, deux envois.

Comme on peut le constater sur la figure 8, on a pour deux envois :

$$C_{\max} = x_1 + p_1 + p_2 + y_1 = x_1 + \sum_{i=1}^{m} p_i + y_1.$$

Et donc :

$$C_{\max} = x_1 + P + d.$$

Cette formule reste applicable dans le cas de plusieurs envois à un seul processeur (voir *fig.* 9 et 10). Par conséquent minimiser x_1 revient à minimiser $C_{\max}$. Nous allons donc chercher à exprimer x_1 en fonction des paramètres P, α et d. Pour cela nous étudions le cas de 2 envois, 3 envois, 4 envois, enfin nous généralisons la formule qui détermine x_1.

Quand on a deux envois (*fig.* 8), on a le système linéaire suivant :

$$x_1 = \alpha p_1 + d,$$

$$x_2 = \alpha p_2 + d,$$

$$p_1 + p_2 = P \text{ (équation de conservation)}$$

$$p_1 = x_2,$$

$$p_2 = z.$$

Il vient donc les équations suivantes :

$$p_1 = \frac{1}{\alpha}(x_1 - d), \tag{r1}$$

$$p_2 = \frac{1}{\alpha}(x_2 - d), \quad \text{or} \quad p_1 = x_2 \text{ donc :} \tag{r2}$$

$$p_2 = \frac{1}{\alpha}\left(\frac{1}{\alpha}(x_1 - d) - d\right) = \frac{1}{\alpha^2}(x_1 - d(1+\alpha)). \tag{r2}$$

Utilisons l'équation de conservation $p_1 + p_2 = P$, il vient :

$$P = \frac{1}{\alpha}(x_1 - d) + \frac{1}{\alpha^2}(x_1 - d(1+\alpha)) = x_1\left(\frac{1}{\alpha} + \frac{1}{\alpha^2}\right) - d\left(\frac{2}{\alpha} + \frac{1}{\alpha^2}\right).$$

D'où :

$$\alpha^2 P = x_1(1+\alpha) - d(1+2\alpha),$$

et enfin :

$$x_1 = \frac{\alpha^2 P + d(1+2\alpha)}{1+\alpha}, \qquad \text{pour } m = 2. \tag{f}$$

Quand on a trois envois (*fig.* 9), le système linéaire est augmenté de l'équation :

$$x_3 = \alpha p_3 + d.$$

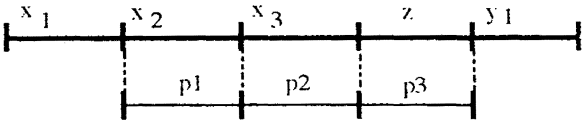

Figure 9. – Cas linéaire, un processeur, trois envois.

L'équation de conservation devient :

$$P = p_1 + p_2 + p_3.$$

Et on a de plus :

$$p_2 = x_3, \qquad p_3 = z.$$

En résolvant selon le même schéma que précédemment, il vient en plus de (r1) et (r2) :

$$p_3 = \frac{1}{\alpha}\left(\frac{1}{\alpha}\left(\frac{1}{\alpha}(x_1 - d) - d\right) - d\right) = \frac{1}{\alpha^3}(x_1 - d(1 + \alpha + \alpha^2)). \qquad \text{(r3)}$$

De même en utilisant l'équation de conservation on obtient :

$$P = \frac{1}{\alpha}(x_1 - d) + \frac{1}{\alpha^2}(x_1 - d(1 + \alpha)) + \frac{1}{\alpha^3}(x_1 - d(1 + \alpha + \alpha^2)),$$

$$P = x_1\left(\frac{1}{\alpha} + \frac{1}{\alpha^2} + \frac{1}{\alpha^3}\right) - d\left(\frac{3}{\alpha} + \frac{2}{\alpha^2} + \frac{1}{\alpha^3}\right),$$

$$\alpha^3 P = x_1(1 + \alpha + \alpha^2) + -d(1 + 2\alpha + 3\alpha^2).$$

Donc :

$$x_1 = \frac{\alpha^3 P + d(1 + 2\alpha + 3\alpha^2)}{1 + \alpha + \alpha^2}, \quad \text{pour } m = 3. \qquad \text{(f)}$$

Quand on a quatre envois (*fig.* 10), le système linéaire est augmenté de l'équation :

$$x_4 = \alpha p_4 + d.$$

L'équation de conservation devient :

$$P = p_1 + p_2 + p_3 + p_4.$$

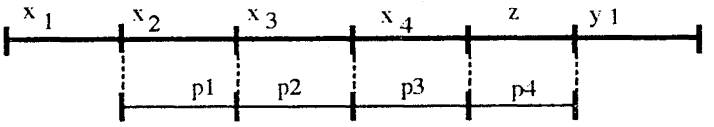

Figure 10. – Cas linéaire, un processeur, quatre envois.

Et on a de plus :

$$p_3 = x_4, \qquad p_4 = z.$$

En résolvant selon le même schéma que précédemment, il vient en plus de (r1), (r2) et (r3) :

$$p_4 = \frac{1}{\alpha^4}(x_1 - d\,(1 + \alpha + \alpha^2 + \alpha^3)). \tag{r4}$$

De même en utilisant l'équation de conservation on obtient :

$$P = x_1\left(\frac{1}{\alpha} + \frac{1}{\alpha^2} + \frac{1}{\alpha^3} + \frac{1}{\alpha^4}\right) - d\left(\frac{4}{\alpha} + \frac{3}{\alpha^2} + \frac{2}{\alpha^3} + \frac{1}{\alpha^4}\right).$$

Donc :

$$x_1 = \frac{\alpha^4 P + d(1 + 2\alpha + 3\alpha^2 + 4\alpha^3)}{1 + \alpha + \alpha^2 + \alpha^3}, \quad \text{pour } m = 4. \tag{f}$$

Nous allons essayer de dégager des formules de récurrence. Regardons tout d'abord la façon dont se décompose le système linéaire :

Équations de base	Conservation	Topologie de la solution
$x_1 = \alpha p_1 + d$ $x_2 = \alpha p_2 + d$ $x_3 = \alpha p_3 + d$ ... $x_m = \alpha p_m + d$	$P = p_1 + \ldots + p_m$	$p_1 = x_2$ $p_2 = x_3$ ... $p_{m-1} = x_m$ $p_m = z$

Intéressons-nous à la première partie de ce système :

$$p_1 = \frac{1}{\alpha}(x_1 - d), \tag{r1}$$

$$p_2 = \frac{1}{\alpha^2}(x_1 - d(1 + \alpha)), \tag{r2}$$

$$p_3 = \frac{1}{\alpha^3}(x_1 - d(1 + \alpha + \alpha^2)), \tag{r3}$$

$$p_4 = \frac{1}{\alpha^4}(x_1 - d(1 + \alpha + \alpha^2 + \alpha^3)), \tag{r4}$$

...

$$p_m = \frac{1}{\alpha^m}(x_1 - d(1 + \alpha + \alpha^2 + \alpha^3 + \ldots + \alpha^{m-1})). \tag{rm}$$

Ensuite grâce à l'équation de conservation il vient :

$$\sum_{i=1}^{m} p_i = P = x_1\left(\frac{1}{\alpha} + \frac{1}{\alpha^2} + \frac{1}{\alpha^3} + \frac{1}{\alpha^4} + \ldots + \frac{1}{\alpha^m}\right) - d\left(\frac{m}{\alpha} + \frac{m-1}{\alpha^2} + \frac{m-2}{\alpha^3} + \frac{m-3}{\alpha^4} + \ldots + \frac{1}{\alpha^m}\right).$$

Donc :

$$P = \frac{1}{\alpha^m}(x_1(1 + \alpha + \alpha^2 + \ldots + \alpha^{m-1}) - d(1 + 2\alpha + 3\alpha^2 + \ldots + m\alpha^{m-1})),$$
$$\alpha^m P = x_1(1 + \alpha + \alpha^2 + \ldots + \alpha^{m-1}) - d(1 + 2\alpha + 3\alpha^2 + \ldots + m\alpha^{m-1}).$$

Et finalement :

$$x_1 = \frac{P\alpha^m + d(1 + 2\alpha + 3\alpha^2 + 4\alpha^3 + \ldots + m\alpha^{m-1})}{1 + \alpha + \alpha^2 + \alpha^3 + \ldots + \alpha^{m-1}}. \tag{f}$$

Donc :

$$x_1 = \frac{P\alpha^m + d\sum_{i=1}^{m} i\alpha^{i-1}}{\sum_{i=1}^{m} \alpha^{i-1}}. \tag{f}$$

Sachant que :

$$C_{\max} = x_1 + P + d.$$

On peut calculer $C_{\max}$, qui à P, d et α donnés dépend de x_1 et donc du nombre d'envois m. On a, par conséquent, la possibilité de déterminer le nombre d'envois m qui donnera un $C_{\max}$ minimum.

Le tableau suivant donne les différentes valeurs de $C_{\max}$ en fonction des autres paramètres et de m.

Ces formules permettent de calculer, en fonction de P, α et d la valeur optimale du nombre d'envois et les valeurs des envois correspondants.

TABLEAU I
Valeur de C_{max} pour un volume de données P, en fonction des paramètres α et d pour différentes valeurs de m.

N° de courbe	α	d	P	m	1	2	3	4	5	6	7	8	9	10
1	1	100	100		400	400	433	475	520	567	614	663	711	760
2	1	70	100		340	325	343	370	400	432	464	498	531	565
3	1	50	100		300	275	283	300	320	342	364	388	411	435
4	1	20	100		240	200	193	195	200	207	214	223	231	240
5	1	10	100		220	175	163	160	160	162	164	168	171	175
6	1	5	100		210	163	148	143	140	139	139	140	141	143
7	1	1	100		202	152	136	129	124	121	119	118	117	117
8	1	0,1	100		200	150	134	125	120	117	115	113	112	111
9	1	1	500		1002	753	670	629	604	588	576	568	562	557
10	10	1	500		1002	957	954	955	956	957	958	959	960	961
11	0,1	1	500		1002	548	507	503	502	502	502	502	502	502

TABLEAU II
Valeur de C_{max} pour un volume de données P, en fonction des paramètres α et d pour différentes valeurs de m (suite).

N° de courbe	α	d	P	m	11	12	13	14	15	16	17	18	19	20
1	1	100	100		809	858	908	957	1007	1056	1106	1156	1205	1255
2	1	70	100		599	633	668	702	737	771	806	841	875	910
3	1	50	100		459	483	508	532	557	581	606	631	655	680
4	1	20	100		249	258	268	277	287	296	306	316	325	335
5	1	10	100		179	183	188	192	197	201	206	211	215	220
6	1	5	100		144	146	148	150	152	154	156	158	160	163
7	1	1	100		116	116	116	116	115	116	116	116	116	116
8	1	0,1	100		110	109	108	108	108	107	107	107	106	106
9	1	1	500		552	549	546	544	542	541	539	538	537	537
10	10	1	500		962	963	964	965	966	967	968	969	970	971
11	0,1	1	500		502	502	502	502	502	502	502	502	502	502

On a reporté, pour 11 valeurs des paramètres P, α et d, les valeurs de C_{max} en fonction de m sur les tableaux I et II et dessiné les courbes correspondantes.

On peut se demander à partir de quel moment il devient intéressant de faire plus d'un envoi de données. Pour cela comparons l'équation correspondant à un envoi avec l'équation correspondant à deux envois, en cas d'équivalence

des deux solutions on a :

(1) $x_1 = \frac{\alpha^2 P + d(1+2\alpha)}{1+\alpha} = d + \alpha P$, puis en simplifiant :

(2) $P = d$.

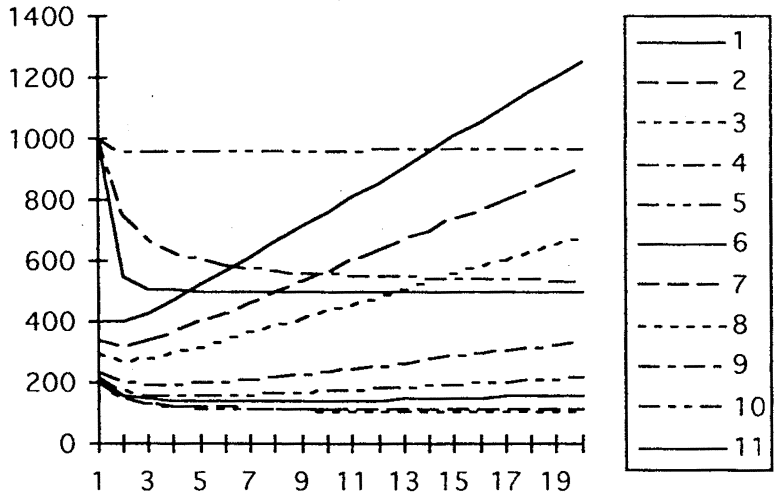

Figure 11. – $C_{\max}$ **en fonction de** m.

Donc si P est plus grand ou égal à d, envoyer deux paquets de données à un seul processeur devient intéressant, sinon il vaut mieux envoyer un seul paquet de données. Cela est relativement logique : si l'on veut bénéficier d'un effet pipeline au niveau macroscopique de l'application, il faut que la quantité à calculer soit plus grande que le temps minimum nécessaire à l'établissement de l'effet pipeline.

7. DEUX PROCESSEURS RECEVANT UN SEUL ENVOI

Nous considérons ici le cas de deux processeurs recevant un seul envoi. Les $g_j(p_i)$ sont, comme dans le cas du paragraphe 6, des fonctions linéaires de la forme : $g_j(p_i) = p_i$. On peut montrer que, du fait que les $g_j(p_i)$ sont linéaires, la propriété 5 reste valable pour le cas d'un bus non critique. Ceci permettra d'écrire les équations. Nous verrons également comment traiter le cas du bus critique.

Quatre signatures sont à étudier : 1212, 2121, 1221, 2112. Par exemple, la première signature (1212) s'interprète de la façon suivante : le premier

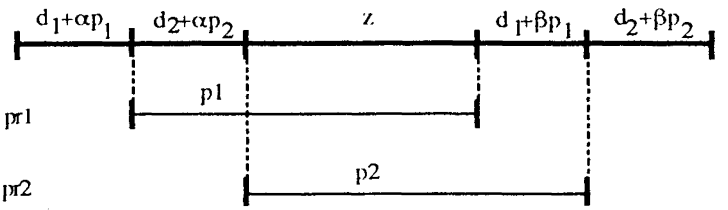

Figure 12. – Cas linéaire, deux processeurs, cas 1212.

envoi est pour le processeur 1, le deuxième, pour le processeur 2, enfin on rapatrie les résultats du processeur 1 avant ceux du processeur 2.

Première signature 1212 :

La figure 12 décrit la solution. On a donc le système d'équations :

(1) $p_1 = z + d_2 + \alpha p_2,$

(2) $p_2 = z + d_1 + \beta p_1,$

(3) $p_1 + p_2 = P$ (avec les inconnues p_1, p_2, z).

Après résolution on obtient :

$$p_1 = \frac{(1+\alpha)P + d_2 - d_1}{2+\alpha+\beta}, \quad p_2 = \frac{(1+\beta)P - d_2 + d_1}{2+\alpha+\beta},$$

$$z = \frac{(1-\alpha\beta)P - (1+\beta)d_2 - (1+\alpha)d_1}{2+\alpha+\beta}.$$

Le temps total d'occupation du bus est $C_{\max} = 2d_1 + 2d_2 + (\alpha+\beta)P + z$. Si $z \leq 0$, alors le bus est critique et on a en fait $C_{\max} = 2d_1 + 2d_2 + (\alpha+\beta)P$ (*fig.* 13).

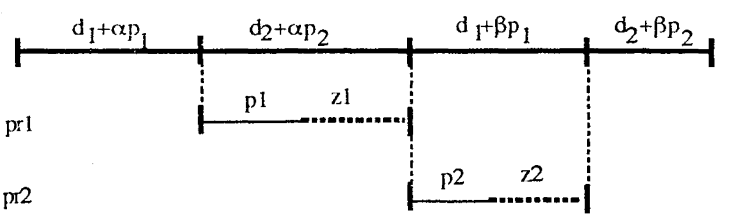

Figure 13. – Cas linéaire, deux processeurs, cas 1212 avec bus critique.

Ce cas n'est pas intéressant car il est dominé par deux envois successifs au processeur le plus proche, c'est-à-dire que l'on revient au cas étudié au paragraphe précédent. On gagne de l'ordre de d_2.

On peut remarquer que, quand P tend vers l'infini, $\frac{p_1}{p_2}$ tend vers $\frac{1+\alpha}{1+\beta}$.

Deuxième signature 2121 :

Ce cas est symétrique du premier en inversant p_1 et p_2 ainsi que d_1 et d_2.

Troisième signature 1221 :

La figure 14 décrit la solution. On a donc le système d'équations :

(1) $p_1 = z + 2d_2 + (\alpha + \beta)p_2$,

(2) $p_2 = z$,

(3) $p_1 + p_2 = P$ (avec les inconnues p_1, p_2, z).

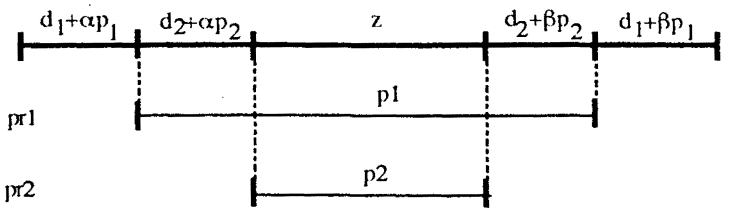

Figure 14. – Cas linéaire, deux processeurs, cas 1221.

Après résolution on obtient :

$$p_1 = \frac{(1+\alpha+\beta)P + 2d_2}{2+\alpha+\beta}, \quad z = p_2 = \frac{P - 2d_2}{2+\alpha+\beta}.$$

Le temps total d'occupation du bus est $C_{\max} = 2d_1 + 2d_2 + (\alpha+\beta)P + z$.

On peut remarquer que, quand P tend vers l'infini, $\frac{p_1}{p_2}$ tend vers $1+\alpha+\beta$.

On peut remarquer également que le cas d'un bus critique est absurde car le processeur 2 n'est pas utilisé. On se retrouve dans le cas où le processeur le plus proche reçoit un seul envoi.

On peut déterminer la valeur de P à partir de laquelle il devient intéressant d'utiliser les deux processeurs. Quand l'utilisation du seul processeur le plus proche est équivalente à l'utilisation des deux processeurs on a : $2d_1 + (1+\alpha+\beta)P = 2d_1 + 2d_2 + (\alpha+\beta)P + z$. On a l'égalité pour $P = 2d_2$ car $z = 0$. Si P est plus grand que cette valeur alors il devient

intéressant d'utiliser les deux processeurs, s'il est inférieur, il vaut mieux utiliser un seul processeur.

Quatrième signature 2112 :

Ce cas est symétrique du troisième en inversant p_1 et p_2 ainsi que d_1 et d_2.

DÉFINITION 6 : On définit le Speed up comme le rapport $\frac{P}{C_{\max}}$.

Nous constatons que le Speed up tend vers 2, le nombre de processeurs, et est croissant avec P.

Nous allons maintenant appliquer ces formules pour des valeurs particulières des paramètres afin de comparer les quatre signatures et de calculer le speed up en fonction de P pour la signature optimale.

TABLEAU III
Répartition des données envoyées aux deux processeurs.

Solution 1212			
P	p_1	p_2	$C_{\max}$
100	76	24	511
5000	2630	2370	3293
10000	5237	4763	6210
15000	7844	7156	9128
20000	10450	9550	12045
25000	13057	11943	14962
30000	15663	14337	17880
35000	18270	16730	20797
40000	20877	19123	23714

Solution 2121			
P	p_1	p_2	$C_{\max}$
100	28	72	511
5000	2583	2417	3291
10000	5190	4810	6208
15000	7796	7204	9126
20000	10403	9597	12043
25000	13009	11991	14960
30000	15616	14384	17877
35000	18223	16777	20795
40000	20829	19171	23712

Solution 1221			
P	p_1	p_2	$C_{\max}$
100	100		511
5000	2773	2227	3277
10000	5403	4597	6197
15000	8033	6967	9117
20000	10663	9337	12036
25000	13294	11706	14956
30000	15924	14076	17875
35000	18555	16445	20795
40000	21185	18815	23715

Solution 2112			
P	p_1	p_2	$C_{\max}$
100	100		511
5000	2725	2275	3325
10000	5355	4645	6244
15000	7986	7014	9164
20000	10616	9384	12083
25000	13246	11753	15003
30000	15877	14123	17923
35000	18507	16493	20842
40000	21137	18863	23762

Pour cela on calcule, à signature fixée, la charge des deux processeurs en fonction de P.

Les coefficients α et β ont été fixés à : $\alpha = 0.1$ et $\beta = 0.01$, de plus $d_1 = 100\, d_2 = 150$. On a donc des distances assez proches.

On peut remarquer que la signature 2112 est toujours dominée par toutes les autres signatures. Ceci s'explique par le fait que l'on a $d_2 > d_1$. On montre facilement que, soit le premier envoi, soit le dernier rapatriement, est pour le processeur le plus proche. Cette propriété se généralise à N processeurs [BCA 94]. On peut remarquer aussi que pour $P = 100$, il est préférable d'utiliser le processeur le plus proche uniquement. En effet, dans ce cas $C_{\max} = 2d_1 + (1 + \alpha + \beta)P = 311 < 511$ (valeur de $C_{\max}$ en utilisant les deux processeurs).

Dressons maintenant un tableau comparatif des temps d'occupation du bus pour les différentes solutions optimales.

TABLEAU IV
Récapitulatif.

P	Sol 1212	Sol 2121	Sol 1221	Sol 2112	Speed up
100	511	511	511	511	0.20
5000	3293	3291	3277	3325	1.53
10000	6210	6208	6197	6244	1.61
15000	9128	9126	9117	9164	1.64
20000	12045	12043	12036	12083	1.66
25000	14962	14960	14956	15003	1.67
30000	17880	17877	17875	17923	1.68
35000	20797	20795	20795	20842	1.68
40000	23714	23712	23715	23762	1.69

Il y a un basculement. Jusqu'à une quantité de données de 30000 la meilleure solution est 1221 puis à partir de 35000 la solution 2121 devient meilleure.

On peut également remarquer que la charge des deux processeurs est équilibrée pour les distances testées.

8. GÉNÉRALISATIONS

Le temps de calcul dépend en général du traitement particulier effectué sur l'envoi i, c'est-à-dire de l'algorithme employé. Une façon de le prendre

en compte est de supposer l'application prédécoupée en k tâches de volumes de données P_1, $P_2, \ldots, P_k$ correspondant à k algorithmes distincts. Notons pour l'envoi i, p_{i1}, $p_{i2}, \ldots, p_{ik}$, les volumes des données des k tâches correspondant à des fractions de données des k algorithmes. De ce fait, g_j serait alors une fonction multilinéaire :

$$g_j(p_{i1}, p_{i2}, \ldots, p_{ik}) = \sum_{p_{ir} \neq 0} (\lambda_{jr} p_{ir} + \omega_{jr})$$

avec

$$\sum_{i=1}^{m} p_{ir} = P_r \qquad (r = 1..k).$$

En fait les résultats se généralisent sans difficulté. Cette première généralisation est très intéressante dans la mesure où elle correspond au cas où les tâches ne sont pas identiques. La première hypothèse importante est le fait que le temps global de calcul ne dépend pas du découpage, la deuxième hypothèse, la possibilité d'un découpage aussi fin que l'on veut. La première hypothèse est réaliste, la deuxième aussi, du moins jusqu'à un grain raisonnable des envois.

Une deuxième généralisation d'intérêt théorique est de se placer dans le cas continu. Nous introduisons une fonction continue $h(x)$ définie sur l'intervalle $[0, P]$ et mesurant la densité du rapport temps de calcul sur volume de données. On aurait alors à déterminer des fonctions continues $\rho_i(x)$ (proportion pour l'envoi i) définies également sur $[0, P]$ telles que :

$$p_i = \left[\int_0^P \rho_i(x) h(x) dx \right] \quad \text{avec} \quad \sum_{i=1}^{m} \rho_i(x) = 1$$

pour tout $x \in [0, P]$ et $g_j(p_i) = \lambda_j p_i + \omega_j$.

Les résultats restent valables. Pour le démontrer il suffit d'approximer les intégrales par des fonctions en escalier.

Remarquons par contre que si les g_j ne sont pas linéaires, les propriétés 3, 4 et 5 ne sont plus valides. Cela voudrait dire que le temps de calcul serait fonction du découpage et pourrait donc se dilater quand on retarde l'envoi d'un morceau à un processeur (comme dans la démonstration de la propriété 3). Si on fait un tri sur un grand ensemble de données la complexité globale dépend du découpage, donc ce cas n'est pas irréaliste.

9. CONCLUSION

Nous avons, dans cet article, mis en évidence un certain nombre de propriétés intéressantes de l'ordonnancement optimal d'une application sur une architecture bus.

Si l'on veut travailler en communications masquées, la politique optimale consiste en général à envoyer la quantité de données juste utile entre deux envois successifs au même processeur. De cette façon ce dernier sera toujours actif et l'utilisation du bus sera optimisée. Ce résultat est doublement intéressant. En effet, il permet de dégager une règle de gestion des tâches. De plus, dans certains cas il permet de résoudre optimalement le problème. Ainsi, pour le cas de plusieurs envois à un seul processeur, et pour le cas d'un seul envoi à deux processeurs et sous l'hypothèse de proportionnalité du temps de calcul, nous avons pu déterminer de façon exacte les quantités à envoyer aux processeurs. Enfin, on peut remarquer que l'étude de plusieurs envois à un seul processeur fait apparaître un effet pipeline au niveau macroscopique d'une application.

Ces résultats théoriques ont été exploités pratiquement pour mettre au point une méthode arborescente déterminant le nombre optimal de processeurs actifs et le nombre d'envois ainsi que leur taille [BCA 94]. Par ailleurs, nous envisageons de généraliser cette approche en étudiant d'autres architectures.

RÉFÉRENCES

[BET 89] D. P. Bertsekas et J. N. Tsitsiklis, Parallel and distributed computation: numerical methods, Prentice Hall, 1989.

[BCA 94] J. P. Boufflet et J. Carlier, An exact method for optimal decomposition and scheduling of an application on a bus-oriented master slaves multiprocessor system, *Fourth International Workshop on Project Management and Scheduling*, July 12-15, 1994, Leuven, Belgium.

[BCS 91] A. Billionnet, M. C. Costa et A. Sutter, Les problèmes de placement dans les systèmes distribués, *TSI*, 1991, *10*, n° 5.

[CAC 88] J. Carlier et P. Chrétienne, Problèmes d'ordonnancement, collection Études et Recherche en Informatique, Masson Éditeur, 1988.

[CHR 92] P. Chrétienne, Ordonnancement et parallélisme, in [CNR92], p. 297-312.

[CNR 92] M. Cosnard, M. Nivat et Y. Robert, Algorithmique parallèle, collection Études et Recherche en Informatique, Masson Éditeur, 1992.

[COT 93] M. Cosnard et D. Trystram, Algorithmes et architectures parallèles, collection Informatique Intelligence Artificielle, Inter Editions Editeur, 1993.

[DAR 92] A. Darte et Y. Robert, Séquencement des nids de boucles, in [CNR 92], p. 343-368.

[ESC 91] Y. Escaig, Une méthode de décomposition de domaine : la sous-structuration, Rapport de thèse, Université de Technologie de Compiègne, 1991.

[FRA 92] P. FRAIGNAUD, Communications dans un réseau de processeurs, in [CNR 92], p. 133-148.

[MUN 91] Méthodes de placement statique des processus sur des architectures parallèles, Traian Muntean et El-Ghazali Talbi, rapport C3, 1991.

[TAB 90] D. TABAK, Multiprocessors, Prentice Hall series in computer engineering, 1990.