UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

Algorithmes parallèles pour DyALog

Isabelle Debourges
Université d'Orléans, LIFO

Rapport N^o **99-04**

Table des matières

Résumé

Since about ten years, Eric Villemonte de la Clergerie (INRIA [1]) has been building a generalised **syntactic analysor**: DyALog.

This software has undergone many evolutions since its birth. But nowadays, it has one serious weakness: response time in the case of ambiguous grammars is much too long.

So, the first goal of the parallelism study we propose for DyALog is a speed-up for execution in the case of Natural Langage grammars which are strongly ambiguous.

This report describes, a **complete** study of possible solutions we could use: 8 algorithms are finally proposed, compared using the **BSP model** [VALL90] [COLL96-1], to obtain a portable prevision of time costs. One of them gives good theorical results for any type of grammar (deterministic or ambiguo us). Four others give good results too, in the case of Natural Langage grammars.

Key words: algorithms, parallelism, BSP, grammars, syntactic analysor, compiler.

1. National Institute of Computer Science and Automatic Research

Résumé

Depuis une petite dizaine d'années, Eric Villemonte de la Clergerie (INRIA) travaille sur la réalisation d'un **analyseur syntaxique** généralisé: DyALog.

Ce logiciel, qui a connu de nombreuses évolutions depuis sa naissance souffre encore d'un handicap: son temps de réponse dans le cas de grammaires ambiguës de grande taille reste trop important.

L'étude de parallélisation de DyALog que nous présentons ici a donc pour but premier d'apporter une solution pour accélérer l'exécution de ce logiciel afin de l'utiliser pour des grammaires du Langage Naturel.

Nous avons, pour cela, réalisé une étude **complète** des solutions envisageables: 8 algorithmes rédigés selon le **modèle BSP** [VALL90] [COLL96-1] sont proposés, et ce afin de garantir une bonne prévision des coûts.
L'un d'entre eux annonce de bons résultats pour tous les types de grammaires (déterministes ou ambiguës). Quatre autres sont très prometteurs pour les grammaires de type Langage Naturel.

Mots Clés: Algorithmes, Parallelisme, BSP, grammaires, analyseur syntaxique, compilateur.

Chapitre 1

Introduction

DyALog, l'analyseur syntaxique généralisé d'Eric de la Clergerie [VILL93], peut à partir d'une grammaire, valider ou réfuter l'appartenance d'une phrase à cette grammaire. Capable de traiter des grammaires hors contexte (CFG) ou de clauses définies (DCG), de petite ou grande taille, déterministes ou fortement ambiguës, il lui reste cependant un point faible: son temps de réponse est exponentiel en la taille du problème. C'est en vue d'une application de DyALog à des grammaires de Langue Naturelle (toujours très complexes) que nous avons fait cette étude de parallélisation, afin de réduire substantiellement les temps d'exécution.

1.1 DyALog séquentiel.

DyALog est à la fois un compilateur et une machine abstraite pour l'exécution tabulaire de programmes et grammaires logiques. DyALog est donc non seulement un analyseur syntaxique, mais aussi un compilateur de programmes logiques. C'est à dire que pour une grammaire fournie en entrée (sous forme de clauses de Horn), DyALog est capable de répondre à une requête, mais aussi de valider ou non une affirmation. On aura rapidement une intuition sur la complexité de ce calcul dès qu'on aura pris conscience que "comprendre une phrase, c'est saisir l'ensemble des connections qui unissent les différents mots" [GUIL93].

En première approximation, DyALog utilise un algorithme de point fixe ascendant avec tabulation et test de subsomption [1]. Ce calcul de point fixe forme la partie centrale de DyALog.

Un algorithme ascendant, ou Bottom-Up part des problèmes les plus simples et utilise les réponses à ceux-ci pour résoudre les problèmes d'ordre supérieur. De plus, mémoriser dans la table le sous-problème généralisé, au sens de la subsomption, au lieu du sous-problème lui-même permet une meilleure réutilisation des sous-calculs: la tabulation permet d'éviter la résolution de certains sous-problèmes quand nous nous rendons compte qu'ils sont instance d'un calcul déjà tabulé. DyALog possède un modèle d'exécution uniforme basé sur la notion d'objet. Un objet est soit un item (à une ou deux composantes) soit une transition de la grammaire. Initialement, l'agenda contient le texte (ou programme) à étudier, la table des objets contient les règles de la grammaire. L'algorithme termine lorsque l'agenda est vide. La boucle centrale de DyALog est illustrée par la figure 1.1. Son exécution se déroule en quatre temps:

1. Au départ de la boucle de contrôle, un objet $\mathcal{O}_1$ est sélectionné dans l'agenda.

1. On dit que t_1 généralise ou subsume t_2, ou que t_2 est une instance de t_1 quand il existe une substitution σ telle que $t_2 = t_1\sigma$

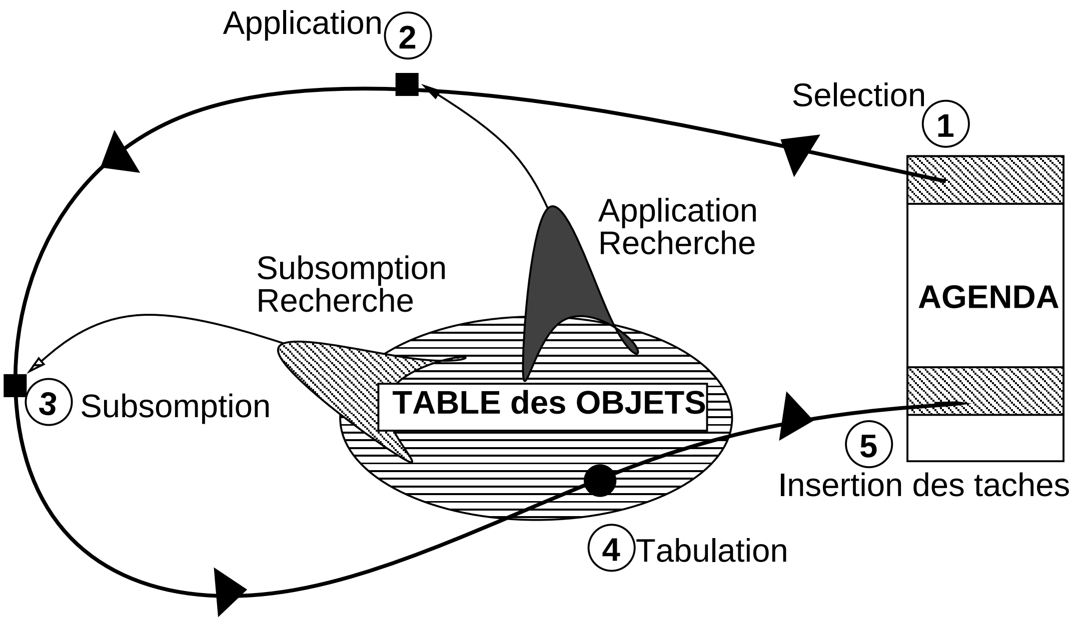


FIG. 1.1 - *Le modèle d'exécution de DyALog séquentiel.*

2. Les objets "applicables" sur $\mathcal{O}_1$ sont alors recherchés dans la table des objets. Chaque objet $\mathcal{O}_2$ ainsi trouvé est appliqué[2] sur $\mathcal{O}_1$ pour produire un nouvel objet $\mathcal{O}_3$ (qui peut être soit un item soit une transition dérivée).

3. Chaque objet $\mathcal{O}_3$ ainsi produit est soumis à un test de subsomption qui se décompose en deux phases distinctes:
   - le test de subsomption faible élimine $\mathcal{O}_3$ s'il est instance d'un objet déja tabulé.
   - le test de subsomption forte élimine les objets déja tabulés instances de $\mathcal{O}_3$. (Dans la pratique, ce test n'est pas appliqué.)

4. Chaque objet nouvellement produit $\mathcal{O}_3$ qui survit au test de subsomption faible est installé dans la table et dans l'agenda.

Remarque: DyALog n'est pas un simple compilateur Prolog. Il a l'immense intérêt de ne pas s'aventurer dans les éventuelles branches infinies de la grammaire, et donc de toujours terminer son exécution.

2. Règle: Un sélectionné ne s'applique à un tabulé que si ce dernier a déja été sélectionné.

Chapitre 2

Evaluation des inefficacités de DyALog séquentiel.

Afin d'évaluer de façon encore plus précise les faiblesses de DyALog séquentiel, et de définir les étapes du calcul à paralléliser, nous avons commencé par faire des mesures sur des coûts particuliers du logiciel.

Nous avons cherché à regarder tout particulièrement:

1. Nombre mini/moyen/maxi d'objets présents dans l'agenda à un instant t.
2. Temps moyen mis pour chaque étape de la boucle générale: recherche d'un élément applicable, application, test de subsomption faible, rangement d'un élément dans la table des objets et l'agenda.
3. L'importance de chaque étape d'un tour de boucle par rapport au temps total d'exécution.

Pour chaque type de grammaire, nous donnons quelques valeurs considérées des plus représentatives. Au niveau des temps d'exécution[1], il n'est pas possible d'accéder à un bon reflet de la réalité si on se contente d'une unique grammaire: on en donne donc plusieurs[2]. Il en est tout autrement pour le nombre des objets présents dans l'agenda: les courbes d'un même type de grammaire sont semblables: on en fournit donc une seule.

1. exprimés en secondes
2. les grammaires utilisées sont disponibles sur ftp://ftp.inria.fr/INRIA/Projects/Atoll/Eric.Clergerie/DyALog/ dans dyalog/Examples

2.0.1 Sur des grammaires du type Langage Naturel.

	gate_CFG_169			
	temps	% temps	nb appels	temps/appel
Temps total	53.51			
$[C]$ sélection	0.76	1.4	19 462	$4 * 10^{-5}$
$[R]$ unification	48.67	90.95	20 075	$2.4 * 10^{-3}$
$[A]$ application	0	0	34 350	0
$[S]$ subsomption	0.68	1.25	18 181	$3.7 * 10^{-5}$
$[I_t]$ ajout tdo	2.38	4.5	20 296	$1.2 * 10^{-4}$
$[I_a]$ ajout agenda	0.06	0.1	19 462	$3 * 10^{-6}$

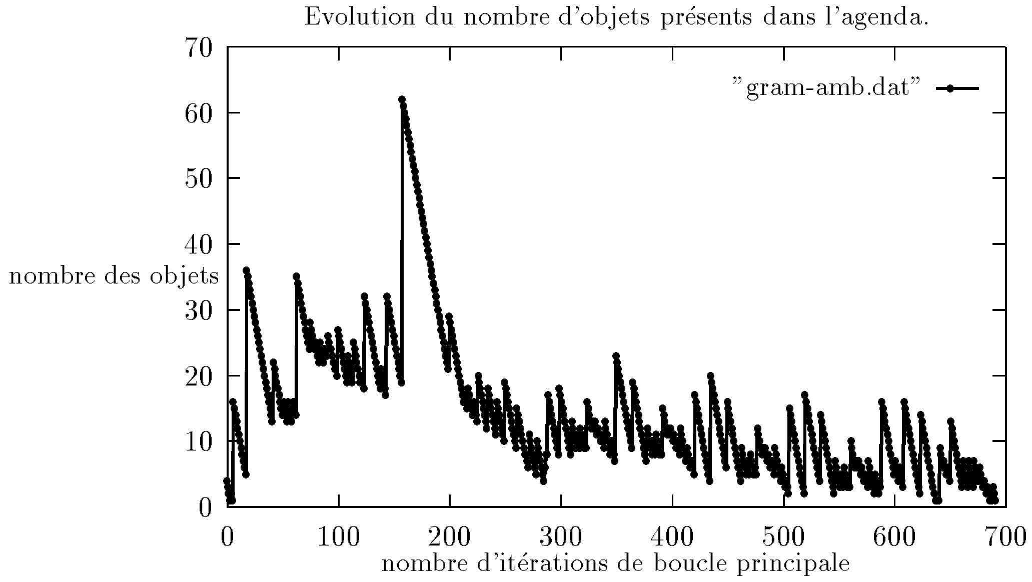
	gate_DCG_169			
	temps	% temps	nb appels	temps/appel
Temps total	65.72			
$[C]$ sélection	0.74	1.1	21 838	$3.4 * 10^{-5}$
$[R]$ unification	56.54	86	21 838	$2.6 * 10^{-3}$
$[A]$ application	0	0	36 317	0
$[S]$ subsomption	0.61	0.9	19 573	$3 * 10^{-5}$
$[I_t]$ ajout tdo	5.42	8.2	22 630	$2.4 * 10^{-4}$
$[I_a]$ ajout agenda	0.09	0.1	21 838	$4 * 10^{-6}$

	tag_3000			
	temps	% temps	nb appels	temps/appel
Temps total	32.01			
$[C]$ sélection	0.45	1.4	16 001	$2.81 * 10^{-5}$
$[R]$ unification	12.83	40.1	21 001	$6 * 10^{-4}$
$[A]$ application	0	0	15 001	0
$[S]$ subsomption	3.39	10.6	10 000	$3.4 * 10^{-4}$
$[I_t]$ ajout tdo	11.94	37.3	19 012	$6 * 10^{-4}$
$[I_a]$ ajout agenda	0.04	0.125	16 001	$2.5 * 10^{-6}$

	dyck_151			
	temps	% temps	nb appels	temps/appel
Temps total	105.83			
$[C]$ sélection	3.04	2.9	12 162	$2.5 * 10^{-4}$
$[R]$ unification	88.4	83.5	18 091	$4.9 * 10^{-3}$
$[A]$ application	0.79	0.75	352 049	$2.3 * 10^{-6}$
$[S]$ subsomption	6.87	6.5	164 160	$4.2 * 10^{-5}$
$[I_t]$ ajout tdo	1.4	1.3	12 319	$1.2 * 10^{-4}$
$[I_a]$ ajout agenda	0.04	0.05	12 162	$3.3 * 10^{-6}$

	bio_123			
	temps	% temps	nb appels	temps/appel
Temps total	219.45			
$[C]$ sélection	3.3	1.5	57 370	$5.7 * 10^{-5}$
$[R]$ unification	196.99	89.8	91 079	$9.8 * 10^{-4}$
$[A]$ application	0	0	356 960	0
$[S]$ subsomption	6.42	2.9	175 445	$3.6 * 10^{-5}$
$[I_t]$ ajout tdo	5.47	2.5	57 510	$9.5 * 10^{-5}$
$[I_a]$ ajout agenda	0.23	0.1	57 370	$4 * 10^{-6}$

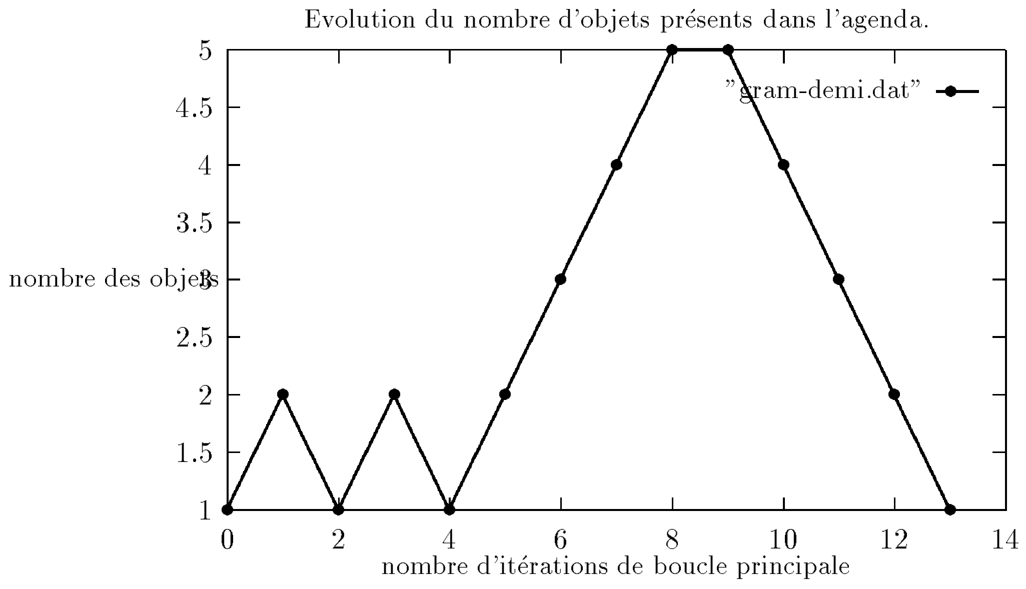
Evolution du nombre d'objets présents dans l'agenda.



2.0.2 Sur des grammaires faiblement ambiguës.

	append(X,Y,[a,b,c,d,e])			
	temps	% temps	nb appels	temps/appel
Temps total	4.49			
$[C]$ sélection	0	0	14	0
$[R]$ unification	0	0	20	0
$[A]$ application	4.45	99.1	18	0.24
$[S]$ subsomption	0	0	6	0
$[I_t]$ ajout tdo	0.01	0.2	24	$4.1 * 10^{-4}$
$[I_a]$ ajout agenda	0	0	14	0

	Hanoï_18			
	temps	% temps	nb appels	temps/appel
Temps total	83.07			
$[C]$ sélection	0.08	0	41	$2 * 10^{-3}$
$[R]$ unification	77.21	93	99	0.8
$[A]$ application	0.18	0.2	138	$1.3 * 10^{-3}$
$[S]$ subsomption	0.01	0	58	$1.7 * 10^{-4}$
$[I_t]$ ajout tdo	0.02	0	86	$2.3 * 10^{-4}$
$[I_a]$ ajout agenda	0	0	41	0



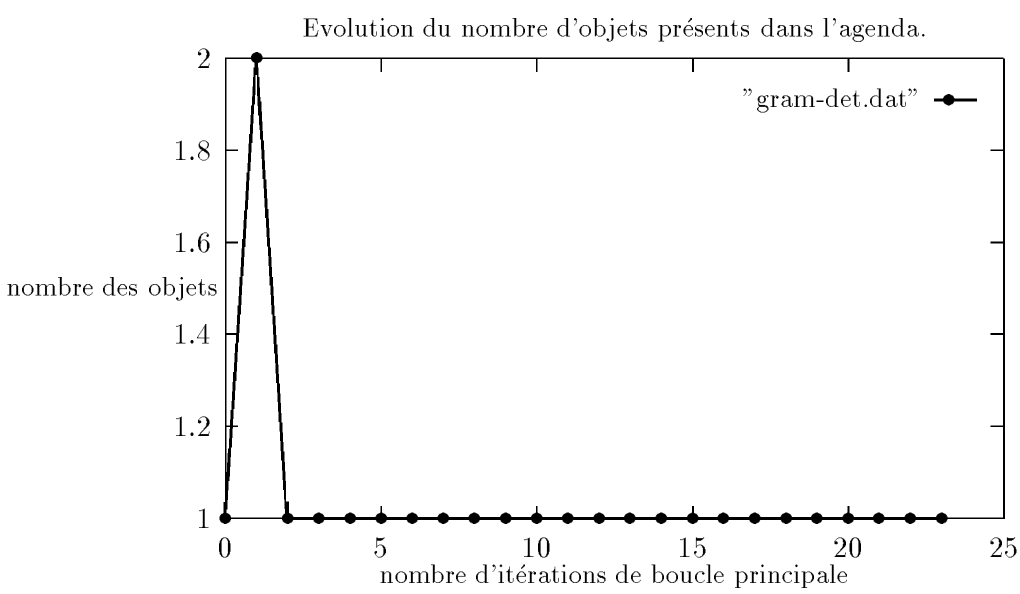
2.0.3 Sur des grammaires déterministes.

	append3000			
	temps	% temps	nb appels	temps/appel
Temps total	10.25			
$[C]$ sélection	0.13	1.27	3 004	$4.3 * 10^{-5}$
$[R]$ unification	0	0	3 005	0
$[A]$ application	8.66	84.5	3 003	$2.8 * 10^{-3}$
$[S]$ subsomption	0	0	1	0
$[I_t]$ ajout tdo	0.80	7.80	3 009	$2.6 * 10^{-4}$
$[I_a]$ ajout agenda	0	0	3 004	0

	append6000			
	temps	% temps	nb appels	temps/appel
Temps total	22.79			
$[C]$ sélection	0.51	2.24	6 004	$8.5 * 10^{-5}$
$[R]$ unification	0	0	6 005	0
$[A]$ application	19.51	85.6	6 003	$3.2 * 10^{-3}$
$[S]$ subsomption	0	0	1	0
$[I_t]$ ajout tdo	1.66	7.28	6 009	$2.7 * 10^{-4}$
$[I_a]$ ajout agenda	0.02	0.08	6 004	$3.7 * 10^{-6}$

	palin_400			
	temps	% temps	nb appels	temps/appel
Temps total	112.63			
$[C]$ sélection	2.84	2.52	81 405	$3.5 * 10^{-5}$
$[R]$ unification	23.95	21.2	81 405	$2.9 * 10^{-4}$
$[A]$ application	19.48	17.3	162 006	$1.2 * 10^{-4}$
$[S]$ subsomption	21.72	19.3	81 002	$2.6 * 10^{-4}$
$[I_t]$ ajout tdo	11.54	10.2	81 812	$1.4 * 10^{-4}$
$[I_a]$ ajout agenda	0.3	0.2	81 405	$3.7 * 10^{-6}$

	palin_200			
	temps	% temps	nb appels	temps/appel
Temps total	22.92			
[C] sélection	0.74	3.22	20 705	$3.6 * 10^{-5}$
[R] unification	5.21	22.73	41 208	$1.2 * 10^{-4}$
[A] application	4.64	20.24	41 006	$1.1 * 10^{-4}$
[S] subsomption	3.82	16.66	20 502	$1.8 * 10^{-4}$
[I_t] ajout tdo	2.21	9.64	20 912	$1.05 * 10^{-4}$
[I_a] ajout agenda	0.16	0.7	20 705	$7.72 * 10^{-6}$



2.1 Les résultats observés et les leçons à tirer.

2.1.1 Sur des grammaires complexes du type Langage Naturel.

Cette catégorie de tests est la plus représentative des problèmes qui prennent un temps d'exécution important: c'est l'ensemble des problèmes qui nous ont amenés à chercher une parallélisation de DyALog. Il semble donc logique de prêter une attention toute particulière à cette catégorie de résultats.

Les tests faits dans cette catégorie ont été réalisés sur des grammaires de type langage naturel de petite et grande taille (de 3 à 103 clauses), très ambiguës.

1. **Les objets dans l'agenda:** cette catégorie de tests nous montre des agendas qui contiennent jusqu'à plusieurs milliers d'objets. Leur nombre ondule au cours de l'exécution: on a une augmentation progressive du cardinal de l'agenda, puis des variations jusqu'à une baisse progressive pour arriver à la fin de l'exécution.

2. **La répartition du temps d'exécution:** c'est la phase d'unification qui est la plus coûteuse en temps (de 40% à 90% du temps d'exécution). Puis les temps d'insertion des objets dans la table des objets et des tests de subsomption prennent respectivement la seconde et la

troisième place avec 1% à 40% et 1% à 10%.
Les temps pris par la sélection, l'ajout dans l'agenda et l'application sont quant à eux souvent négligeables. (Le temps d'application est dans la plupart des cas nul).

De plus... Pour ce type de grammaires, on remarque que:

1. les environnements sont petits (donc peu de fiches)
2. le nombre des variables par fiche est restreint

2.1.2 Sur des exemples simples non déterministes.

Cette catégorie de tests est représentée par des problèmes comme les n reines (nqueens), les tours de Hanoï, les grands-pères dans le cas d'une famille à ramifications ...

1. **Les objets dans l'agenda:** le nombre d'objets présents dans l'agenda ne stagne pas. Sa courbe présente un pic atteint à la suite de variations, puis elle descend régulièrement jusqu'à la fin de l'exécution.
2. **La répartition du temps d'exécution:** on ne peut pas être très catégorique sur la répartition de celui-ci. Les valeurs obtenues sont très changeantes d'une grammaire à l'autre.

2.1.3 Sur des exemples simples déterministes.

1. **Les objets dans l'agenda:** pour la plupart des tests effectués, le nombre des objets présents dans l'agenda tout au long de l'exécution reste à 1. Pour les rares cas où ce n'est pas le cas, il ne grandit que très peu (jusqu'à 3) au cours de l'exécution pour vite redécroître. *Il faut donc penser qu'une parallélisation du type algorithme I[3] (ou une de ses évolutions) n'aura pas d'impact sur les coûts en temps de telles exécutions.*
2. **La répartition du temps d'exécution:** le temps pris par l'application arrive toujours en tête de course (c'est exactement l'opposé des grammaires très ambiguës). Quant aux autres, tout dépend de la grammaire. La sélection, les insertions des objets dans la table des objets et l'agenda ne sont que peu coûteux.

3. cf page 10

AIDE MEMOIRE.

Généalogie des algorithmes parallèles proposés pour DyALog :

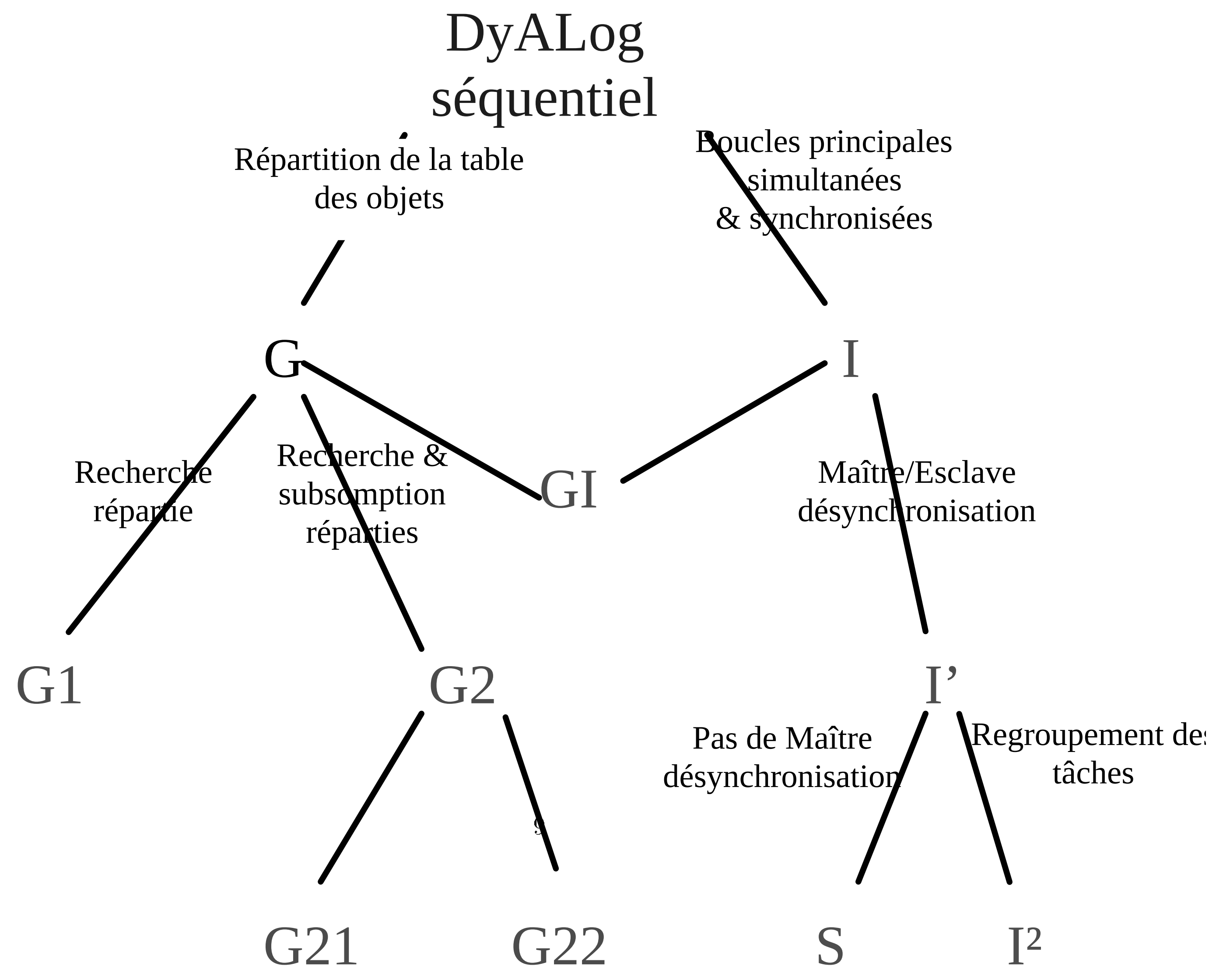


FIG. 2.1 - *Généalogie des algorithmes parallèles proposés.*

Chapitre 3

Les idées approfondies.

3.1 Introduction.

Les contraintes imposées par la structure de données de DyALog nous ont obligés à éliminer des solutions potentielles. Pourtant après un tel balayage, il reste encore des possibilités. Nous présentons ici les solutions survivantes.

L'arbre généalogique des algorithmes proposés, présent page 9 vous facilitera la lecture du present chapitre.

3.1.1 Correction et complétude de l'étude.

Dans la suite de ce chapitre, nous présentons 8 algorithmes parallèles pour DyALog. Une telle quantité peut paraître dénuée de sens par sa grandeur. Mais il n'en est rien. Chacun des algorithmes présente un intérêt particulier et trouve donc sa place dans ce document. En effet l'étude de parallélisation de DyALog que nous présentons ici se doit d'être **correcte** mais aussi **complète**.

Nous avons, dans le chapitre “Bourse aux idées de parallélisation” présenté les trois types de parallélisme existants: parallélisme de données, de contrôle, de flux.

Le parallélisme de données est applicable dans le cas de DyALog. En effet, on peut imaginer que les processeurs traitent la tâche correspondant à un objet O_1 dans différentes parties de la table des objets. Cette idée donnera naissance à une première famille d'algorithmes.
Le parallélisme de flux n'est pas adapté à notre problème. En effet, on ne peut pas définir de vecteur de données auquel on ferait subir une séquence de traitements.

Il reste le parallélisme de contrôle qui, lui aussi, s'adapte sans trop de difficultés à notre problème. C'est ce type de parallélisation que nous allons utiliser pour créer une deuxième famille d'algorithmes.

Maintenant que nous connaissons les types de parallélisme à mettre en oeuvre, il faut mettre en évidence des tâches indépendantes dans l'exécution de DyALog séquentiel. Deux voies s'offrent à nous:

- Les recherches des objets unifiables à un objet sélectionné, dans des parties distinctes de la table des objets sont indépendantes. Il en est de même pour le test de subsomption ($\Rightarrow$

parallélisme de données).

- Deux tours de boucle principale de DyALog sont (quasi) indépendants ($\Rightarrow$ parallélisme de contrôle).

On peut donc distinguer deux idées directrices qui permettent de classer ces algorithmes:

1. *les algorithmes où les processeurs se partagent la recherche d'unification*: c'est le cas de la solution G1, et de la solution G2.
2. *les algorithmes où les processeurs font des recherches isolées et se communiquent ensuite les résultats*: solution I, ses évolutions: les modèles maître/esclave et la solution S.

L'algorithme hybride de G et I est au confluent de ces deux idées.

L'ensemble des solutions présentées ci-après est implantable sur une **machine BSP**: un ensemble de processeurs à mémoire locale, reliés par un réseau de communication, muni d'un système de synchronisation globale. Ce modèle nous permettra de prévoir les performances des algorithmes proposés.

Pour chaque algorithme, nous donnerons ses spécificités, le détail de l'algorithme, une idée de son exécution, une justification intuitive de la correction de sa sémantique vis à vis de DyALog séquentiel, et sa complexité en moyenne ou dans le pire cas.

3.1.2 Notations utilisées.

Pour les calculs de complexité en moyenne de chaque algorithme, nous utiliserons les notations suivantes:

- p: nombre de processeurs utilisés.
- n: nombre de tours de boucles principales effectués par DyALog séquentiel. C'est aussi le nombre total d'objets passant par l'agenda au cours de l'exécution.
- C: temps moyen nécessaire pour la sélection d'un objet dans l'agenda.
- R: temps moyen nécessaire dans DyALog séquentiel, pour la recherche des objets O_2 applicables à un objet O_1, dans la table des objets.
- A: temps moyen nécessaire pour l'application d'un objet O_2 à un objet O_1 pour donner O_3.
- S: temps moyen nécessaire, dans DyALog séquentiel, pour un test de subsomption faible sur un objet O_3.
- I_t: temps moyen nécessaire à l'insertion d'un objet dans une table des objets.
- I_a: temps moyen nécessaire à l'insertion d'un objet dans un agenda.
- l: paramètre BSP de la machine. Temps requis pour synchroniser tout le système lors d'une communication.
- **g**: paramètre BSP de la machine. Temps nécessaire pour acheminer une 1-relation de communication.

3.1.3 Remarques.

- Les complexités fournies dans ce document sont calculées en considérant que tous les processeurs sont en activité: ils ont tous en charge une partie de la tâche globale à exécuter. Ceci n'est évidemment pas toujours réaliste dans le cas de distribution des objets de l'agenda si le nombre des objets présents dans celui-ci est inférieur au nombre de processeurs [1].

- De plus, un éventuel abus de langage peut être commis: O_2 désigne basiquement un objet applicable à un O_1, mais il pourra accessoirement désigner l'ensemble des objets applicables à O_1.
  Il en est de même pour O_3 qui pourra désigner l'ensemble des objets nés de $O_1 \otimes O_2$.

- Le parametre BSP g utilisé dans les calculs de complexité est valable pour le transport d'un objet: il est exprime en flops/objet (un objet est composé de 5 mots, cf page 58).

- Les barrières de synchronisation ayant lieu lors de chaque communication ne sont pas mentionnées; elles sont sous-entendues.

3.2 Solution G: la recherche à plusieurs.

Chaque processeur dispose d'une mémoire dans laquelle sont copiés l'agenda et la table des objets, avec un système d'adressage global. Chaque processeur dispose, au sein de sa mémoire, de son lot de piles de stockage des opérations en cours.

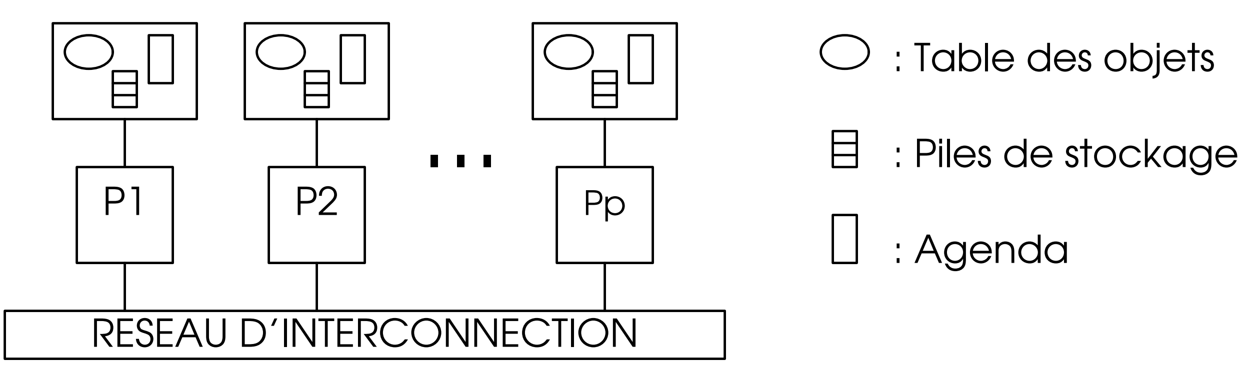


FIG. 3.1 - *Solution G: la machine.*

L'idée fondatrice de cette solution est une **simulation de la table des objets répartie** sur les mémoires des différents processeurs: chaque processeur possède la table complète mais n'en explore qu'une partie au moment de faire une recherche. Cette répartition des recherches entre les processeurs est dynamique.

De cette idée sont nées deux variantes: G1 et G2.

La première, G1, fait répartir la recherche des objets applicables à un même objet O_1 entre les processeurs, puis chaque processeur fait les **tests de subsomption dans toute la table**, sur les

1. en pratique, il faudra s'assurer que l'on n'utilise pas plus de processeurs qu'il n'y a d'objets dans l'agenda

objets candidats qu'il a lui-même créés.

La seconde, G2, fait répartir le travail, entre les processeurs, tout au long de l'algorithme: la recherche des termes applicables est répartie, les objets produits sont communiqués et **les tests de subsomption sont aussi répartis**.

Voici la description de ces deux propositions d'algorithmes.

3.3 G en une étape: solution G1.

L'algorithme s'écrit alors:

Tant que Agenda_Non_Vide **faire**

Par $i = 1$ à p **faire**
[1] sélectionner 1^{er} objet de l'agenda: O_1
dépiler 1^{er} objet de l'agenda
[2] recherche d'unification dans la $i^{ème}$ portion de la table des objets
[3] application des O_2 trouvés à O_1 pour donner les O_3
[4] test de subsomption faible sur les O_3 dans toute la table
[5] envoi des objets O_3 créés aux $(p-1)$ autres processeurs
fin_par

Par $i = 1$ à p **faire**
[6] insertion des objets reçus dans la table des objets
[6] insertion des objets reçus dans l'agenda
fin_par

fin_tant_que

3.3.1 Une idée de l'exécution de G1.

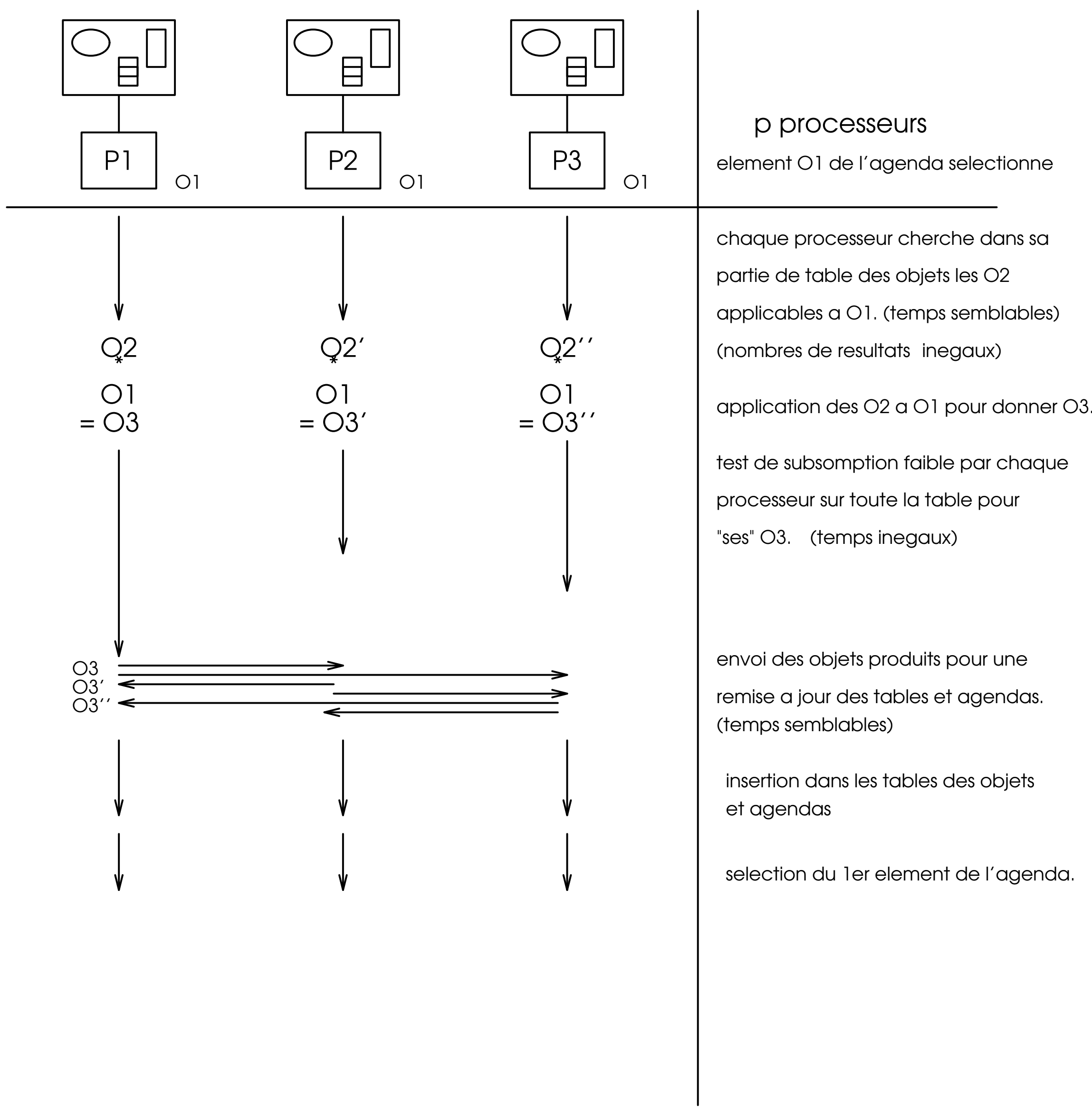


FIG. 3.2 - *Solution G1: exécution.*

3.3.2 Conservation de la sémantique.

Dans cet algorithme, tous les processeurs cherchent simultanément les objets O_2 applicables à un même O_1. Ainsi, obtient-on à la fin d'un tour de boucle, l'ensemble des O_2 contenus dans la table des objets à la fin du tour de boucle précédent. On calcule donc exactement le même ensemble que dans le cas de DyALog séquentiel, où la règle "*un sélectionné ne s'applique à un tabulé que si celui-ci a déja été sélectionné*" est utilisée.

L'ensemble des objets O_3 nés de $O_1 \otimes O_2$ est ensuite engendré. (Résultat trivialement égal à celui obtenu en séquentiel).

L'ensemble des O_3 est ensuite soumis au test de subsomption faible. On peut observer un cas particulier en dehors duquel tout se passe exactement comme en séquentiel:

si $\exists O_2, O_2^{'}$ provenant de processeurs différents tels que:

$$\begin{array}{lcl} O_1 \otimes O_2 & \mapsto & O_3 \\ O_1 \otimes O_2^{'} & \mapsto & O_3^{'} \end{array}$$

avec

$$O_3^{'} \;\; \succeq^{2} \;\; O_3$$

alors le plus général des deux n'est pas éliminé dans l'exécution parallèle réalisée par l'algorithme G1.

Mais qu'en serait-il dans l'exécution séquentielle de DyALog? Deux cas se présentent:

- O_3 est créé avant $O_3^{'}$.
  Comme $O_3^{'}$ est instance de O_3, $O_3^{'}$ est éliminé par le test de subsomption faible.

- $O_3^{'}$ est créé avant O_3.
  Comme O_3 est plus général que $O_3^{'}$, O_3 est inséré dans la table des objets et l'agenda.

La différence d'exécution rencontrée pour O_3 créé avant $O_3^{'}$ devrait être éliminée au sein du modèle séquentiel par le test de subsomption forte. Pourtant celui-ci n'est pas réalisé car il reste trop coûteux par rapport au gain de temps qu'il pourrait engendrer. C'est pourquoi le choix des objets les plus généraux les premiers, dans l'agenda, lui a été préféré.
Le même résultat que celui donné par l'exécution parallèle de G1 peut donc être rencontré, dans une exécution séquentielle et ce dès que l'on change l'ordre de sélection des objets dans l'agenda.

On considérera donc que le cas particulier de l'exécution parallèle exposé ici ne se présente que rarement et ne conduit en aucun cas à un résultat inexact: l'ajout d'un test de subsomption faible et/ou forte entre les objets créés et tabulés n'est donc pas justifié.

L'algorithme G1 a soit la même sémantique que celle de DyALog séquentiel, soit une sémantique cohérente avec celle-ci.

3.3.3 La complexité de G1.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant[3]:

tâche	coût
1	C
2	$\frac{R}{p}$
3	$\frac{x}{p}A$
4	$\frac{x}{p}S$
5	$x'(p-1)g+l$
6	$x'(I_t+I_a)$

2. peut être lu comme “est instance de”
3. notations définies page 11

Avec

x : nombre d'objets O_2 applicables à un O_1.

x': nombre de O_3 créés (après test de subsomption) pour un O_1.

Nota: le facteur $\frac{x}{p}$ ne vaut que si ces x nouveaus objets sont bien répartis...$\frac{x}{p}$ est idéal.

Cette boucle sera répétée n fois au cours de l'exécution. Ce qui nous donne la complexité globale $\mathbf{C_{G1}}$ suivante:

$$\mathbf{C_{G1}} = n(C + \frac{R}{p} + \frac{x}{p}(A + S) + x'(p-1)g + l + x'(I_t + I_a))$$

3.3.4 Remarques.

Les côtés positifs.

- Quel que soit le nombre d'objets présents dans l'agenda, l'accélération sera la même. Ainsi, le nombre d'objets dans l'agenda toujours égal à 1, n'est pas un handicap. Cet algorithme est donc utilisable dans le cas d'applications de DyALog à des problèmes déterministes.
- Aucun problème pour tenir à jour la liste des objets déjà sélectionnés.

Les côtés négatifs.

- Le temps mis par un processeur pour rechercher les objets O_2 applicables à O_1 n'est pas prévisible. On peut pourtant penser que la variance entre les temps de recherche pour un même nombre d'éléments à tester ne sera pas trop importante du fait de la répartition quasi-équitable[4] du nombre des objets dans les différentes portions de table des objets attribuées à chaque processeur. En revanche, ce qui n'est pas prévisible dans cette configuration, c'est la variance entre les nombres d'objets O_3 créés, et par là même le temps demandé par les tests de subsomption faible qui en découlent. (C'est le point que l'algorithme G2 se propose de résoudre).
- Comment connaître la variance entre les cardinaux des ensembles d'éléments O_2 présents dans chaque partie de la table?
  Ce résultat pourrait nous permettre d'estimer l'attente des processeurs avant la synchronisation globale réalisée pour l'envoi des objets dans l'étape 5.

3.4 G en deux étapes: solution G2.

Cet algorithme s'écrit:

---

4. voir à ce sujet page 47

Tant que Agenda_Non_Vide **faire**

Par $i = 1$ à p **faire**

[1] sélectionner 1^{er} objet de l'agenda: O_1
dépiler 1^{er} objet de l' agenda
[2] recherche d'unification dans la $i^{ème}$ portion de la table des objets
[3] application des O_2 trouvés à O_1 pour donner les O_3
fin_par

[4] Test de subsomption faible sur les O_3 créés
(algorithme G21 ou algorithme G22)

Par $i = 1$ à p **faire**

[5] insertion des objets reçus dans la table des objets
[5] insertion des objets reçus dans l'agenda
fin_par

fin_tant_que

Deux possibilités s'offrent à nous dans la façon de réaliser le test de subsomption en répartissant la table des objets entre les processeurs:

- la première possibilité, exploitée par G21, est: une communication des O_3 à tous les processeurs. Les objets O_3 construits sont ensuite pris tour à tour. Tous les processeurs cherchent **simultanément dans des parties distinctes de la table des objets** pour faire subir le test de subsomption au même nouvel objet O_3. Il faut alors: soit un système de communication aussi rapide que possible pour que le test s'arrête dès qu'un des processeurs a trouvé un élément dont le O_3 en cours de test est instance (peu efficace), soit laisser finir le test.

- la seconde possibilité, exploitée par G22, est qu'à chaque étape du test, un processeur teste, dans **sa partie de la table, les objets produits par un même processeur**, qui n'ont pas encore été éliminés par le test. Une fois le test fini, le processeur i envoie les objets O_3 qui ont survécu à son test, au processeur *i+1*. Cette opération doit être effectuée $n - 1$ fois.

Algorithme de subsomption pour G21:

Par $i = 1$ à p **faire**
[1] envoyer les O_3 créés aux $(p-1)$ autres processeurs
[2] stocker les O_3 reçus dans une file $F1$, triés
dans l'ordre des numéros des processeurs dont ils proviennent.
fin_par

Par $i = 1$ à p **faire**
Tant que $F1$_non_vide **faire**
défiler un objet
[3] le soumettre au test de subsomption faible dans la $i^{ème}$ portion de la table.
si l'objet survit au test alors on l'insère dans une file $F2_i$
sinon rien
fin_tant_que
fin_par

[4] calcul de l'intersection[5] des $F2 = \bigcap_{i=1}^{i=p} F2_i$
[5] envoi de[6] $F2$ à tous les processeurs

Algorithme de subsomption pour G22:

Par $i = 1$ à p **faire**
[1] tester les O_3 créés par i dans la $i^{ème}$ portion de la table des objets
[2] Ens_obj $\leftarrow$ objets survivants
fin_par

Pour $k = 1$ à $(p-1)$ **faire**
Par $i = 1$ à p **faire**
[3] envoyer Ens_obj à $i+1$
[4] test de subsomption faible sur les objets reçus
[2] Ens_obj $\leftarrow$ objets survivants
fin_par
fin_pour

Par $i = 1$ à p **faire**
[5] envoi des objets aux $(p-1)$ autres processeurs
fin_par

Une idée des exécutions de G21 et G22.

5. Comme l'intersection est associative, on peut utiliser la réduction binaire équilibrée (cf [GIBB88]). Un algorithme effectuant cette tâche est proposé page 24.
6. On pourrait ici utiliser l'algorithme de diffusion parallèle proposé dans [GIBB88] pour un petit gain de temps supplémentaire.

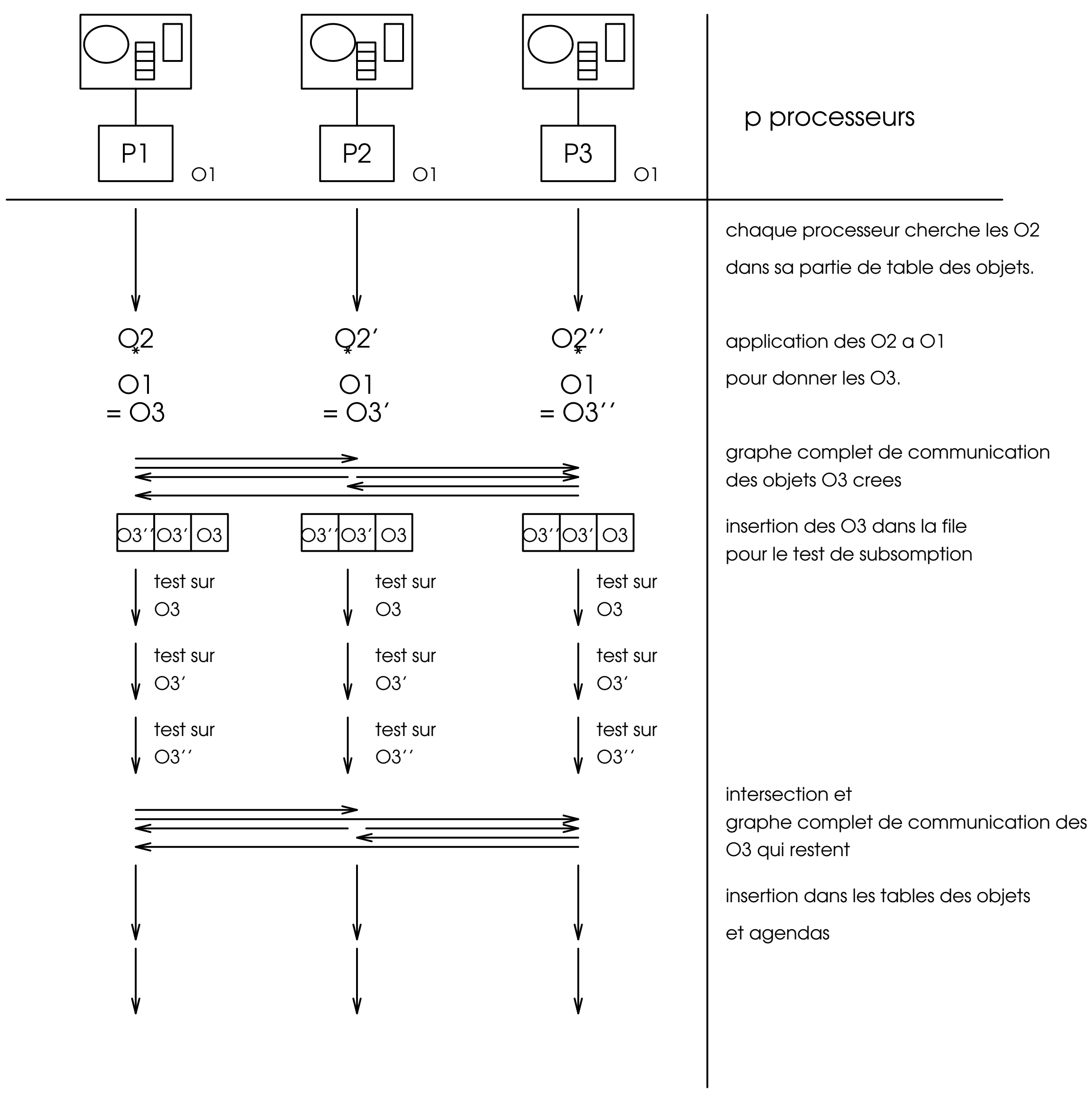


Fig. 3.3 - *Solution G21: exécution.*

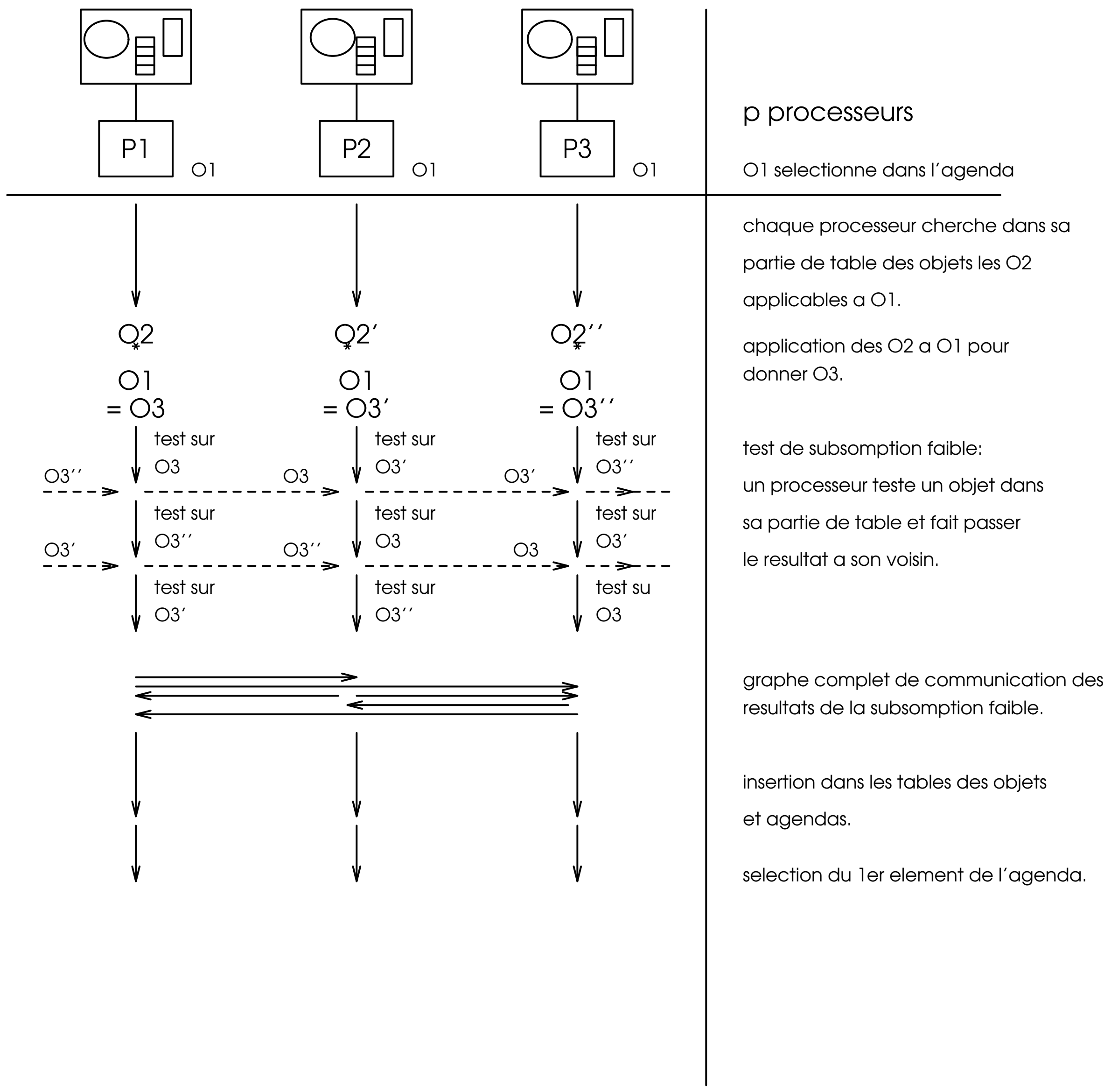


FIG. 3.4 - *Solution G22: exécution.*

3.4.1 Conservation de la sémantique.

Les sémantiques des algorithmes G21 et G22 sont strictement les mêmes que celle de l'algorithme G1. On peut donc faire les mêmes commentaires que ceux faits page 14: la sémantique de DyALog séquentiel est conservée.

3.4.2 Complexités de G2, G21 et G22.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant [7]:

7. les notations sont définies page 11

tâche	coût
1	C
2	$\frac{R}{p}$
3	$\frac{x}{p}A$
4	θ
5	$x'(I_t + I_a)$

Avec
x : nombre d'objets O_2 applicables à un O_1.
x': nombre de O_3 créés (après test de subsomption) pour un O_1.
θ: temps nécessaire au test de subsomption faible; dépend de l'algorithme choisi entre G21 et G22.
Nota: $\frac{x}{p}$ est ici aussi idéal.

Cette boucle sera répétée n fois au cours de l'exécution.
La complexité $\mathbf{C_{G2}}$ de cet algorithme est donc:

$$\mathbf{C_{G2}} = n(C + \frac{R}{p} + \frac{x}{p}A + \theta + x'(I_t + I_a))$$

Le coût θ du test de subsomption faible peut varier en fonction de l'algorithme choisi. Nous allons donc détailler ici le coût de chacun d'eux:

Algorithme G21		Algorithme G22	
tâche	coût	tâche	coût (pire cas)
1	$\frac{x}{p}(p-1)g + l$	1	$\frac{xS}{p^2}$
2	ε	2	ε'
3	$p\frac{x}{p}\frac{S}{p} = \frac{xS}{p}$	3	$(p-1)[\frac{x}{p}g + l]$
4	Π	4	$(p-1)[x\frac{S}{p^2}]$
5	$x'(p-1)g + l$	5	$x'(p-1)g + l$

Avec
x : nombre d'objets O_2 applicables à un O_1.
x': nombre de O_3 créés (après test de subsomption) pour un O_1.
ε, ε': temps nécessaires à la gestion de la file. (négligeables)
Π: temps nécessaire au calcul de l'intersection.

Les complexités des tests de subsomption sont donc:

$$\theta_1 = (p-1)g(\frac{x}{p} + x') + 2l + \frac{x}{p}S + \Pi + \varepsilon \quad (3.1)$$

$$\theta_2 = (p-1)[\frac{x}{p}g + x'g + l] + \frac{x}{p}S + l + \varepsilon' \quad (3.2)$$

Ce qui nous donne les complexités globales $\mathbf{C_{G21}}$ et $\mathbf{C_{G22}}$ respectives suivantes:

$$\begin{aligned}\mathbf{C_{G21}} &= n(C + \frac{R}{p} + \frac{x}{p}A + \theta + x'(I_t + I_a)) \quad (3.3)\\ &= n(C + \frac{R}{p} + \frac{x}{p}(A + S) + (p-1)g(\frac{x}{p} + x') + 2l + \Pi + \varepsilon + x'(I_t + I_a))\end{aligned}$$

$$\begin{aligned}
\mathbf{C_{G22}} &= n(C + \frac{R}{p} + \frac{x}{p}A + \theta + x'(I_t + I_a)) \qquad (3.4)\\
&= n(C + \frac{R}{p} + \frac{x}{p}(A+S) + (p-1)[\frac{x}{p}g + x'g + l] + l + \varepsilon' + x'(I_t + I_a))
\end{aligned}$$

3.4.3 Comparaison des deux algorithmes.

Pour cette étude, nous nous donnons les paramètres suivants supposés connus:

- S: temps moyen nécessaire, dans DyALog séquentiel, pour un test de subsomption faible.
- p': nombre de processeurs ayant un $\eta_{\mathbf{b}} \neq 0$.
- $\eta_{\mathbf{b}}$: nombre total des objets O_2 trouvés au cours d'une itération dans DyALog séquentiel = nombre des O_3 avant test de subsomption faible = x'.
- $\eta_{\mathbf{e}}$: nombre total des objets O_3 ayant survécu au test de subsomption faible, au cours d'une itération.

Cas général.

Pour connaître la meilleure des deux solutions, dans le cas général, il faut résoudre l'inéquation suivante:

$$C_{G21} \leq C_{G22} \qquad (3.5)$$

qui se réécrit en:

$$\Pi \leq (p-2)l \qquad (3.6)$$

avec

$$\begin{aligned}
\Pi &= \log_2 p(\text{communication} + \text{calcul})\\
\Pi &= \log_2 p(x'g + l + x')
\end{aligned}$$

Les valeurs de x pour lesquelles il faut changer d'algorithme dépendent de la machine choisie.

Cas où $p' = 1$: un unique O_3 a été créé sur l'ensemble des processeurs.

Il est utile, dans ce cas, de faire travailler plusieurs processeurs pour effectuer le test de subsomption si et seulement si le coût des communications n'est pas supérieur au gain de temps. (Avec: $x = 1$; seul le processeur qui a créé l'objet doit l'envoyer aux $p-1$ autres processeurs.) C'est à dire qu'il faut:

$$\begin{aligned}
\theta_1 &\leq S\\
2(p-1)g + 2l + \varepsilon + \frac{S}{p} + \Pi &\leq S \qquad (3.7)
\end{aligned}$$

pour l'algorithme 1, ou bien

$$\begin{aligned} \theta_2 &\leq S \\ S + (p-1)(2g+l) + l + \varepsilon' &\leq S \end{aligned} \tag{3.8}$$

pour l'algorithme2.

Il est évident que l'inéquation 3.8 n'admet pas de solution. Le second algorithme n'est donc pas approprié dans ce cas précis: ceci était intuitivement prévisible.

Tandis que 3.7 peut se réécrire en:

$$(p-1)[2g - \frac{1}{p}S] + 2l + \varepsilon + \Pi \leq 0 \tag{3.9}$$

ou encore en

$$p^2[2g] + p[2l - 2g + \log_2 p(x'g + l + x') + \varepsilon - S] + S \leq 0 \tag{3.10}$$

Cas où $var(\eta_b)$ est grande.

Dans ce cas, on sait qu'il existe une très grande probabilité pour qu'il y ait un déséquilibre entre les nombres des objets O_2 produits par les différents processeurs. La question se pose alors sur l'intérêt éventuel d'une redistribution[8] des objets entre les processeurs avant de démarrer le test de subsomption faible.

Algo G21: Un rééquilibrage du nombre des objets sera coûteux en temps, alors que l'exécution qui suivra sera la même que sans $\Rightarrow$ ceci implique une augmentation inutile du coût.

Algo G22: Pour qu'une redistribution des objets soit bénéfique, il faut:

$$\text{coût avec rééquilibrage} \leq \text{coût sans rééquilibrage} \tag{3.11}$$

Après simplifications, le coût sans rééquilibrage s'écrit:

$$(p-1)g(max) + S(max) \tag{3.12}$$

alors qu'avec rééquilibrage:

$$(p-1)g(moy) + S(moy) + \text{équilibrage} \tag{3.13}$$

Où
max = nombre maxi d'objets créés par un même processeur avant équilibrage.
moy = nombre maxi d'objets détenus un même processeur après équilibrage.

On pose $\Delta = max - moy$.
Il faut donc, pour avoir un gain de temps:

$$\Delta \geq \frac{\text{équilibrage}}{(p-1)g + S} \tag{3.14}$$

8. On propose un algorithme pour effectuer cette tâche page 24

3.4.4 Remarques.

Les côtés positifs.

- Tous les processeurs finissent une recherche commune à peu près en même temps: d'où un gain de temps (moins d'attente) pour la synchronisation.(A condition de choisir une répartition dynamique des objets à étudier.)
- Quel que soit le nombre d'objets présents dans l'agenda, l'accélération sera la même. Ainsi, le nombre d'objets dans l'agenda toujours égal à 1, n'est pas un handicap. Cet algorithme est donc utilisable dans le cas d'applications de DyALog à des problèmes déterministes.
- Pas de problème pour tenir à jour la liste des objets déjà sélectionnés.

3.4.5 L'algorithme d'intersection.

L'idée de base est celle de la réduction binaire équilibrée:

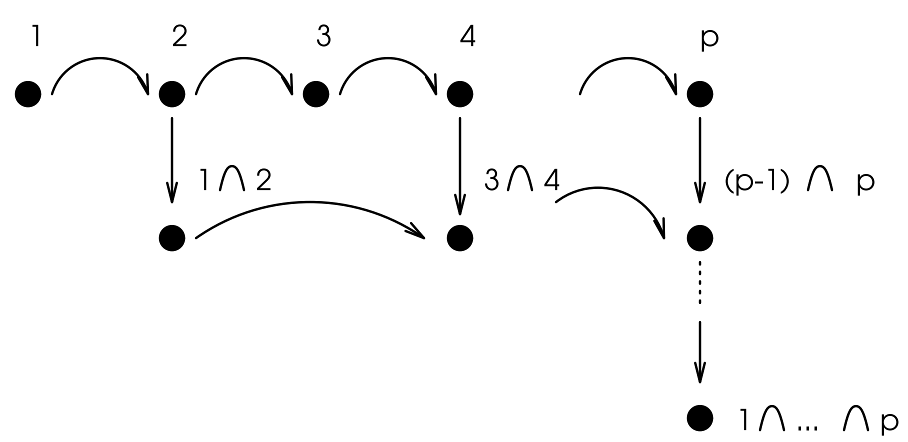


FIG. 3.5 - *Intersection: exécution.*

La complexité s'écrit:

$$\begin{aligned} \Pi &= \log_2 p(\text{communication} + \text{calcul}) \\ &= \log_2 p(x'g + l + x') \end{aligned}$$

Le resultat final du calcul se trouve dans le processeur p.

3.4.6 L'algorithme de redistribution.

On pose:

• $Hist(j)$ = nombre d'objets détenus par le processeur j avant rééquilibrage.

• si $\sum_{i=1}^{p} Hist(j) \bmod p = 0$ alors $Block(j) = \frac{\sum_{j=1}^{j=p} Hist(j)}{p}$

sinon $Block(j) = \sum_{j=1}^{j=p} Hist(j) \bmod p + 1$ pour $j \in \{1.. \sum_{i=1}^{i=p} Hist(j) \bmod p\}$

et $Block(j) = \sum_{j=1}^{j=p} Hist(j) \bmod p$ pour les autres.

• $Rest(j) = Hist(j) - Block(j)$.

• α = nombre de processeurs dont $Rest(j) > 0$.

• β = nombre de processeurs dont $Rest(j) < 0$.

• $Proc_{k=1...\alpha+\beta}$ (k)= le tableau des indices des processeurs dont $Rest(j) \neq 0$ triés par ordre décroissant de $Rest(j)$.

Le principe de l'algorithme est illustre par la figure 3.6

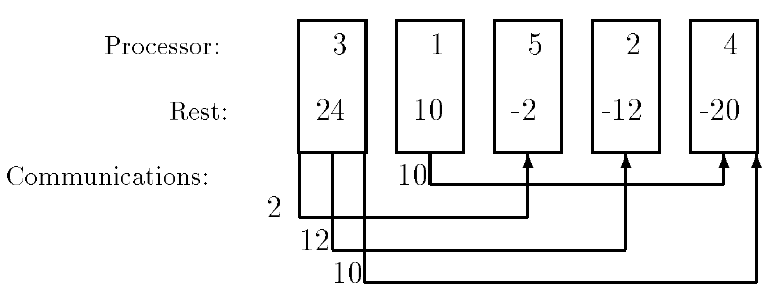


FIG. 3.6 - *Ordres d'envoi de la fonction de rééquilibrage.*

L'algorithme de redistribution[9] s'écrit alors:

$i := 1$; $j := \alpha + 1$;
Tant que $(i \leq \alpha)$ **faire**
  Tant que $Rest_{Proc(i)} > 0$ **faire**
    n_objets:= $min(Rest_{Proc(i)}, -Rest_{Proc(j)})$;
    envoyer$(Proc(i), Proc(j), n_objets)$;
    $Rest_{Proc(i)} := Rest_{Proc(i)}$- n_objets;
    $Rest_{Proc(j)} := Rest_{Proc(j)}$+ n_objets;
    si $Rest_{Proc(i)} = 0$ **alors** $i := i + 1$;
    si $Rest_{Proc(j)} = 0$ **alors** $j := j + 1$;
    fin_tant_que
  fin_tant_que

La complexité de cet algorithme s'écrit:

$$\mathbf{C}_{\mathbf{redistri}} = \sum_{j=1}^{j=p} Hist(j) \times \log_2(p).$$

9. Cet algorithme est fortement inspiré du papier de M.Bamha et G.Hains, *A self-balancing join algorithm for SN machines* à paraître dans PDCS'98

3.5 Solution I: la multiplication des données.

L'idée de cet algorithme est de faire **plusieurs tours de boucles de DyALog séquentiel, en même temps**. Ceci avec une synchronisation globale du système à la fin de chaque tour de boucle parallèle.

Dans ce cas (comme dans la solution G), chaque processeur dispose d'une mémoire dans laquelle sont copiés l'agenda et la table des objets, avec un système d'adressage global. Chaque processeur dispose de son lot de piles de stockage des opérations en cours au sein de sa mémoire.

On suppose dans cet algorithme que l'agenda contient toujours au moins p objets au début de chaque boucle parallèle. Si tel n'est pas le cas (si l'agenda contient $q < p$ objets) alors les q premiers processeurs seront actifs et exécuteront l'algorithme, tandis que les $(p - q)$ autres resteront passifs en attendant le debut du prochain tour de boucle. L'exécution se verra alors un peu ralentie, et par là même, le temps d'exécution augmentera.
Cet algorithme est donc **plus particulièrement adapté aux grammaires fortement ambiguës** (voir page 3).

L'algorithme s'écrit:

Tant que Agenda_Non_Vide **faire**

  Par $i = 1$ à p **faire**
    [1] sélectionner le $i^{ème}$ objet de l'agenda: O_{1_i}
      dépiler p objets de l'agenda
    [2] recherche d'unification sur O_{1_i} dans toute la table des objets
    [3] application des O_{2_i} trouvés à O_{1_i} pour donner les O_{3_i}
    [4] test de subsomption faible sur les O_{3_i} dans toute la table
    [5] envoi des objets O_{3_i} créés aux $(p - 1)$ autres processeurs
    fin_par

  Par $i = 1$ à p **faire**
    [6] insertion des objets reçus dans la table des objets
    [6] insertion des objets reçus dans l'agenda
    fin_par

fin_tant_que

3.5.1 Une idée de l'exécution de I.

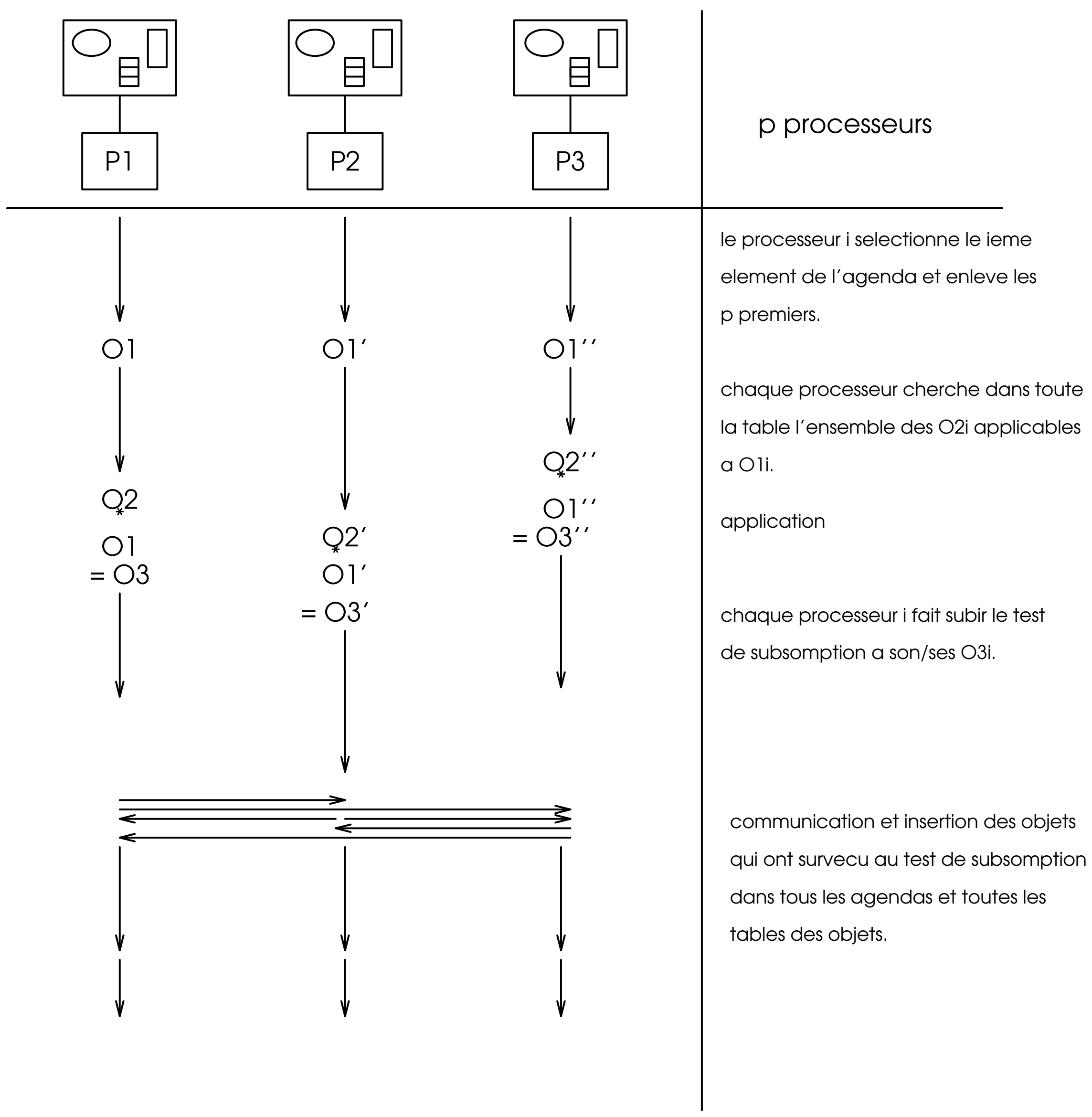


FIG. 3.7 - *Solution I: exécution.*

3.5.2 Conservation de la sémantique.

Pour vérifier que la sémantique de I est la même que celle de DyALog séquentiel, on peut se poser la question suivante: *peut-il exister un O_1 et un O_2 applicables l'un à l'autre et qui ne se rencontreront jamais?*

3 cas se présentent: O_1 sélectionné avant O_2
O_1 sélectionné en même temps que O_2
O_1 sélectionné après O_2.

O_1 sélectionné avant O_2.
O_2 ne sera pas retenu comme unifiable pendant le traitement de la tâche d'agenda correspondante à O_1. (Pourtant O_2 est présent dans la table des objets: mais c'est la règle "*un sélectionné ne s'applique à un tabulé que si celui-ci a déja été sélectionné* " qui l'emporte). Leur unification aura

lieu au moment de la sélection de O_2.
On ne constate donc pas de différence avec l'exécution de DyALog séquentiel.

<u>O_1 sélectionné en même temps que O_2.</u>
Le marquage de la table des objets (dès la sélection et avant le traitement) va faire que:

$$\begin{aligned} O_1 \otimes O_2 &\mapsto O_3 \\ O_2 \otimes O_1 &\mapsto O_3 \end{aligned}$$

Il faut donc éventuellement un test d'égalité au moment de l'insertion des objets dans la table des objets et l'agenda pour pallier une insertion double d'un même objet. (voir si un tel test ne coûtera pas plus cher que deux traitements identiques dont un disparaitra au moment du test de subsomption faible).
La sémantique est donc la même que dans DyALog séquentiel.

<u>O_1 sélectionné après O_2.</u>
O_2 sera retenu comme unifiable à O_1 pendant le traitement de la tâche d'agenda correspondant à O_1. On aura alors:
$O_1 \otimes O_2 \rightarrow O_3$.
Il n'y a donc pas de différence avec la sémantique de DyALog séquentiel.

On ne peut donc pas avoir de O_1 et O_2 applicables l'un à l'autre qui ne se rencontrent jamais. La sémantique de I reste donc la même que celle de DyALog séquentiel.

3.5.3 Complexité de I.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant [10]:

tâche	coût
1	C
2	R
3	xA
4	xS
5	$x'(p-1)g+l$
6	$px'(I_t+I_a)$

Avec
x : nombre d'objets O_2 applicables à un O_1.
x': nombre de O_3 créés (après test de subsomption) pour un O_1.

Nota: Le coût de la première étape est en réalité un petit peu plus important que C: on dépile p objets au lieu d'un seul comme dans la définition de C, mais nous considérerons cette différence comme négligeable.

10. les notations sont définies page 11

Cette boucle sera répétée $\frac{n}{p}$ fois au cours de l'exécution (à condition que l'agenda ait toujours assez d'objets en stock pour faire travailler p processeurs, cad p objets dans sa structure à chaque début de boucle). Ce qui nous donne la complexité globale $\mathbf{C_I}$ suivante:

$$\mathbf{C_I} = \frac{n}{p}(C + R + x(A + S) + x'(p-1)g + l + px'(I_t + I_a))$$

3.5.4 Remarques.

Les côtés positifs.

- Cette solution est directement applicable à la structure séquentielle actuelle de DyALog: elle ne nécessite pas de changement radical de la structure de données.
- Le déroulement de cette solution semble dirigé par un chef d'orchestre: la synchronisation globale. La trace d'exécution sera aisément lisible.
- On peut faire en sorte que toutes les tables des objets soient rangées exactement pareil en insérant les objets dans le même ordre.

Les côtés négatifs.

- Cette solution n'est recevable que si la différence de temps mis entre deux processus simultanés pour effectuer leur tour de boucle n'est pas trop importante.
- Dans le cas où le nombre des objets présents dans l'agenda serait constant, égal à 1 tout au long de l'exécution, aucune accélération ne pourrait être constatée. On pourrait même avoir un ralentissement de l'exécution dû aux communications inter-processeurs.

Implantation. Au moment de la sélection, le processeur i sélectionne le $i^{\grave{e}me}$ objet. Il en dépile p et marque ces p objets comme *déja sélectionnés* dans la table des objets; ceci afin d'assurer la conservation de la sémantique de DyALog séquentiel.

3.6 Evolution de I: solution Maître/Esclave.

Cet algorithme reprend l'idée de plusieurs boucles de DyALog séquentiel réalisées en même temps (idée de l'algorithme I), avec une amélioration du côté des synchronisations. On élimine les synchronisations globales de chaque tour de boucle, et on ajoute **un maître chargé de distribuer**, aux processeurs, **les tâches à exécuter de l'agenda**: on obtient une exécution **totalement asynchrone**.

Cet algorithme (et son évolution I^2) ne devront pas être programmés avec la librairie BSP-lib de BSP; en effet, le modéle BSP est ici uniquement utilisé pour analyser le coût de ces algorithmes.

La mémoire du premier processeur contient l'agenda, tandis que les $p-1$ autres contiennent chacune une copie de la table des objets, avec un système d'adressage global. Chaque processeur (sauf le premier) dispose de son lot de piles et d'une file de stockage des opérations en cours au sein de sa mémoire. On dispose d'un système d'envoi de messages entre les processeurs.

Nous appellerons "serveur" le processeur P_0.

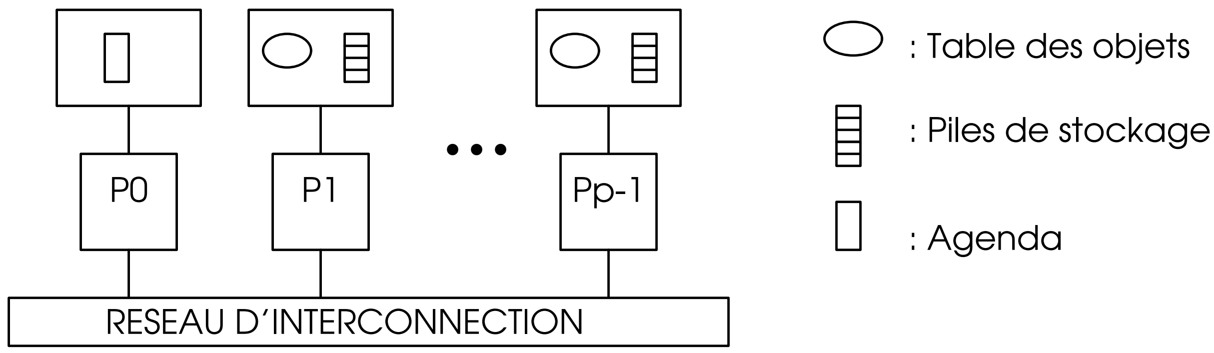


FIG. 3.8 - *Solution Maître/Esclave: la machine.*

Le principe utilisé est appelé **Maître/Esclave**.

L'algorithme est alors décrit par:

Par

Par $i = 1$ **faire**
demande_de_tache $\Rightarrow$ | insérer les objets reçus dans l'agenda
dépiler un objet O_1 de l'agenda
l'envoyer au demandeur

Par $i = 2$ à p **faire**

Tant que non Fini **faire**
[1] demander une tâche au serveur et lui envoyer O_{3_i}
[2] insérer les objets de la file F_i dans la table des objets
vider la file F_i ; réinitialiser O_{3_i}
$O_{1_i} \leftarrow$ tâche reçue du serveur
[3] recherche d'unification sur O_{1_i} dans toute la table des objets
[4] application des O_{2_i} aux O_{1_i} pour donner les O_{3_i}
[5] test de subsomption faible sur les O_{3_i}
$O_{3_i} \leftarrow$ objets survivants
enfiler O_{3_i} dans F_i
[6] envoyer O_{3_i} aux $(p-2)$ autres clients et au serveur
enfiler les objets reçus au cours du traitement dans F_i

si Agenda_Vide et Aucun_Traitement_En_Cours **alors** Fini $\leftarrow$ Vrai

fin_tant_que

fin_par

fin_par

L'algorithme s'arrête quand l'agenda n'a plus de tâche à traiter, et que les processeurs ont tous fini de traiter les leurs.

Pour tenir à jour la liste des objets déjà sélectionnés dans toutes les tables des objets, le serveur doit envoyer avec chaque nouvelle tâche la liste des tâches distribuées depuis la dernière communication: il lui faut un tableau de files pour stocker ces informations spécifiques à chaque esclave. En moyenne, ceci engendre un envoi de p objets à chaque demande d'une nouvelle tâche.

3.6.1 Une idée de l'exécution de la solution Maître/Esclave.

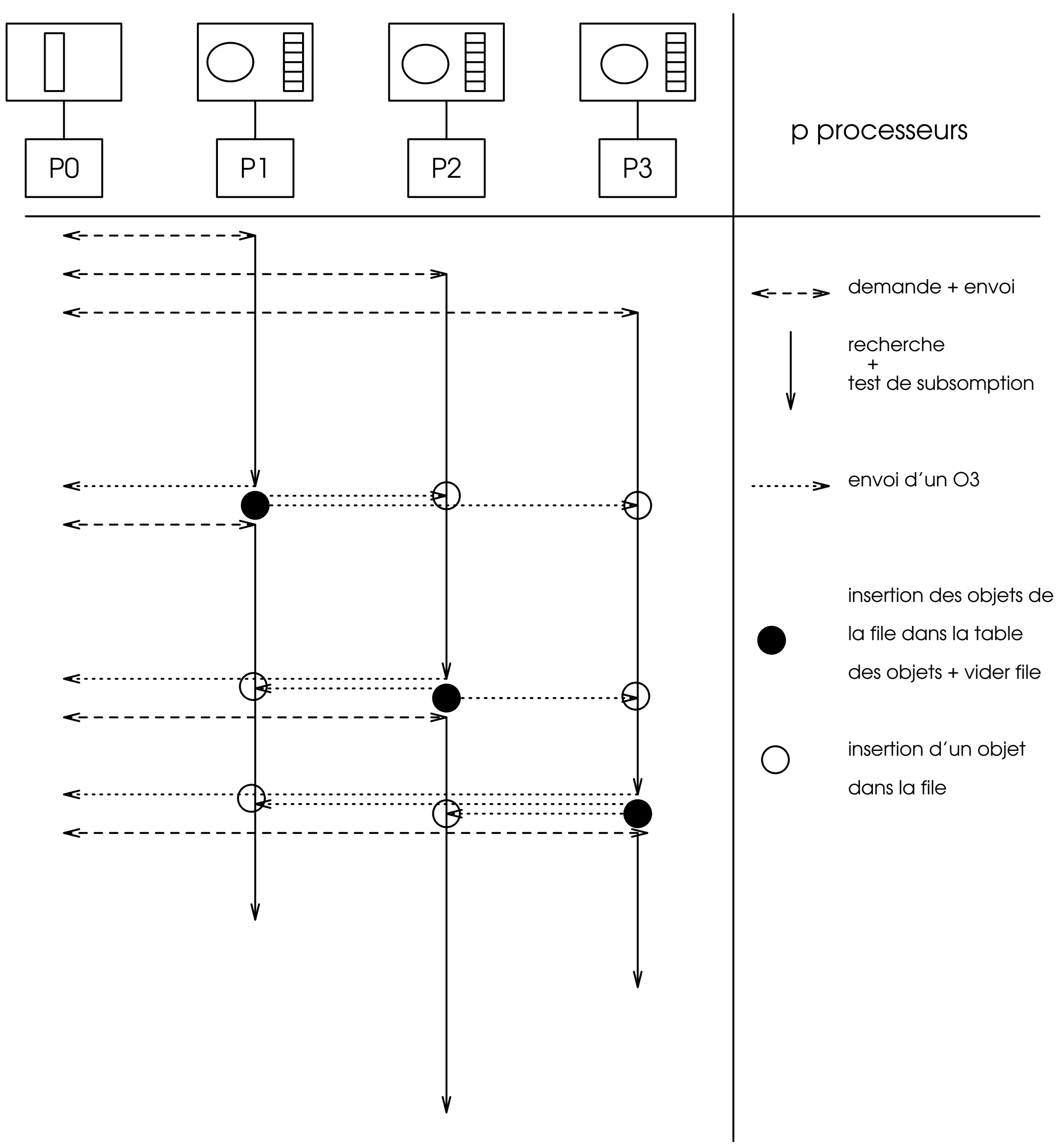


FIG. 3.9 - *Solution Maître/Esclave: exécution.*

3.6.2 Conservation de la sémantique.

Nous allons étudier la question: *Peut-on avoir un O_1 et un O_2 compatibles tels que O_1 et O_2 ne se rencontrent jamais?*

On peut répondre, que dès que l'un des objets (O_1 par exemple) est créé, il est diffusé à tous les processeurs: le Maître et les Esclaves. Il est alors inséré dans les tables des objets avant tout démarrage d'une nouvelle tâche, et peut dès lors être sélectionné dans l'agenda. Deux cas se présentent alors:

1. O_2 existe déjà.
2. O_2 n'existe pas encore.

O_2 existe déjà: alors O_2 est nécessairement dans la table des objets.
Si O_2 a déjà été sélectionné, alors O_1 s'applique à O_2.
S'il n'a pas encore été sélectionné alors O_1 ne s'applique pas à O_2. Ils seront unifiés au moment de la sélection de O_2 dans l'agenda.

O_2 n'existe pas encore: au moment de la sélection de O_2 dans l'agenda, O_1 et O_2 seront unifiés.

On peut donc dire que pour tous O_2 et O_2 compatibles, il n'existe pas de cas défavorable dans lequel ces deux objets ne pourraient donner naissance au O_3 attendu, ce qui aurait pu modifier la sémantique initiale de DyALog.

La sémantique de I' est donc la même que celle de DyALog séquentiel.

3.6.3 Complexité.

L'algorithme I' n'est pas réellement BSP: il est désynchronisé. Il n'est donc pas évident d'évaluer son coût. Pourtant, on peut l'approximer par l'algorithme BSP I'_{BSP} qui s'écrit comme I' avec en plus une synchronisation globale après chaque communication d'un O_{3_i}.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant[11]:

tâche	coût
1	$C + 1g$
2	$(p-1)x'I_t$
3	R
4	xA
5	xS
6	$x'(p-1)g + l$

Avec
x : nombre d'objets O_2 applicables à un O_1.
x': nombre de O_3 créés (après test de subsomption) pour un O_1.

Nota 1: on considère que l'agenda sélectionne la tâche tant que le processeur remet sa table des objets à jour.
Nota 2: on a considéré les temps de gestion de la file comme négligeables.

11. les notations sont définies page 11

Cette boucle sera répétée $\frac{n}{p-1}$ fois au cours de l'exécution. Ce qui nous donne la complexité globale $\mathbf{C_{I'_{BSP}}}$ suivante:

$\mathbf{C_{I'_{BSP}}} = \frac{n}{p-1}((p-1)x'I_t + C + g + R + x(A+S) + x'(p-1)g + l)$

3.6.4 Saturation du serveur.

On sait qu'un tour de boucle esclave coûte:

$$(p-1)x'I_t + C + g + R + x(A+S) + x'(p-1)g + l \tag{3.15}$$

Une communication au serveur coûte: $x'g + l + x'I_a + C + (1+p)g$
(envoi des objets créés + synchro + insertion des objets par le serveur dans son agenda + sélection d'une tâche + envoi de la tâche et des nouveaux objets sélectionnés).

Pour que le serveur ne soit pas au stade de saturation, il faut:

$$1 \text{ tour de boucle esclave} \geq (p-1)\text{communications au serveur} \tag{3.16}$$

Les valeurs acceptables de p vérifient l'équation:

$$-p^2 + p[x'(I_t - I_a) - 2x'g - l - C] + [x'(I_a - I_t) + R + x(A+S) + 2C + 2g + 2l + x'g] \geq 0 \tag{3.17}$$

Elles sont à l'intérieur des racines.

3.6.5 Remarques.

Les côtés positifs.

- Ce modèle semble très intéressant (en termes de gain de temps) dans le cas où les temps de recherche plus test de subsomption ne sont pas semblables.

Les côtés négatifs.

- La trace d'exécution d'un tel algorithme ne sera certainement pas facile à suivre à la main du fait de l'inexistence de synchronisation globale des processeurs dans l'algorithme I'.

- Il existe un nombre maximum de processeurs envisageable au dela duquel le serveur ne sera plus assez rapide: il saturera.

- Ce modèle est valable pour une machine pourvue d'un nombre important de processeurs. A l'opposé, il ne l'est certainement pas pour une machine bi-processeur: ce modèle réquisitionne un processeur qui ne fait que servir les autres.

- Ce modèle peut, dans l'absolu, être plus rapide que I. Pourtant, comme on utilise le calcul des coûts de BSP, on n'arrive pas à un tel résultat à cause des synchronisations obligatoires.

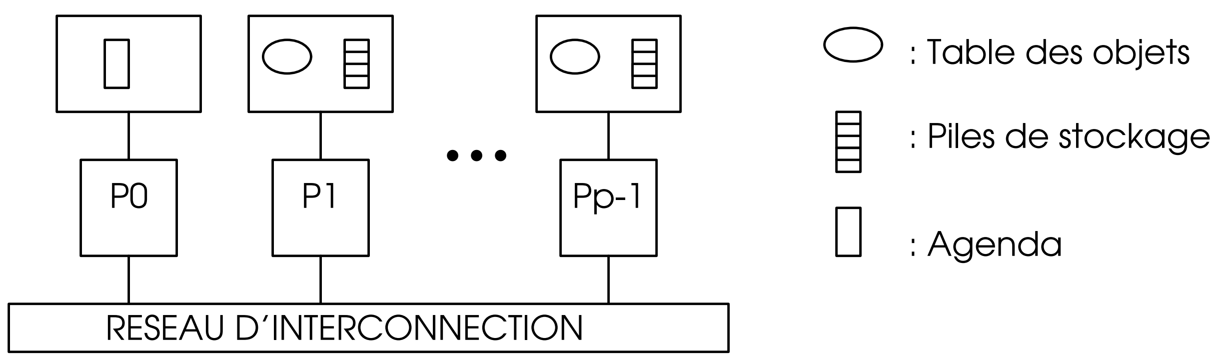


FIG. 3.10 - *Evolution² de I: la machine.*

3.7 Une évolution² de I.

Cet algorithme s'inspire largement du modèle Maître/Esclave décrit précédemment, avec une évolution supplémentaire: le Maître **distribue les tâches par paquets de m**, au lieu de 1 à chaque fois. Le côté **asynchrone** de l'exécution de I' est conservé.

La mémoire du premier processeur contient l'agenda, tandis que les $p-1$ autres contiennent chacune une copie de la table des objets, avec un système d'adressage global. Chaque processeur (sauf le premier) dispose de son lot de piles de stockage des opérations en cours au sein de sa mémoire.

Nous appellerons "serveur" le processeur P_0.

L'algorithme est alors décrit par:

Par

Par $i = 1$ faire
demande_de_taches $\Rightarrow$ insérer les objets recus dans l'agenda
depiler m objets de l'agenda
les envoyer au demandeur

Par $i = 2$ à p **faire**

Tant que non Fini **faire**
[1] envoyer F_i au serveur et demander m tâches au serveur
enfiler les objets reçus au cours du traitement dans F_i
[2] insérer les objets de la file F_i dans la table des objets
vider la file F_i

Pour $j = 1$ à m **faire**
$O_{1_{i,j}} \leftarrow j^{ème}$ objet reçu du serveur
[3] recherche d'unification sur $O_{1_{i,j}}$ dans toute la table des objets
[4] application des $O_{2_{i,j}}$ aux $O_{1_{i,j}}$ pour donner les $O_{3_{i,j}}$
[5] test de subsomption faible sur les $O_{3_{i,j}}$
$O_{3_i} \leftarrow O_{3_i} \bigcup O_{3_{i,j}}$ survivants
fin_pour

enfiler O_{3_i} dans F_i
[6] envoyer O_{3_i} aux $(p-2)$ autres clients

si Agenda_Vide et Aucun_Traitement_En_Cours **alors** Fini $\leftarrow$ Vrai

fin_tant_que

fin_par

fin_par

3.7.1 Une idée de l'exécution de I^2

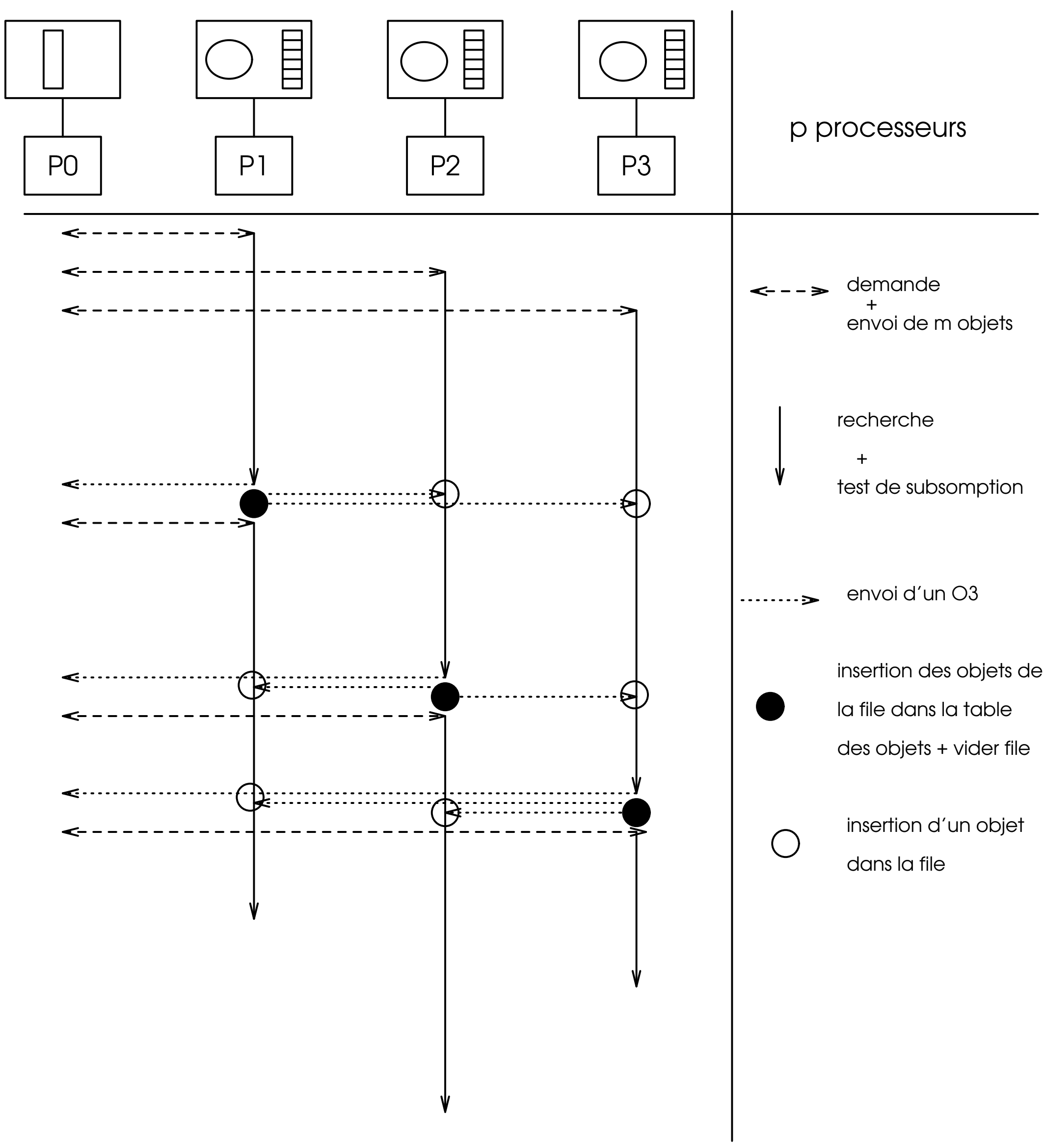


FIG. 3.11 - *Solution I^2: exécution.*

3.7.2 Conservation de la sémantique.

Il ne faut pas perdre de vue que le fait de grouper à l'extrême les tâches, peut changer la sémantique. Il faut alors peut-être, revoir le choix fait dans DyALog séquentiel, qui consistait à ne pas faire le test de subsomption forte, prévu à l'origine.
Nous donnons ici une preuve intuitive de la conservation de la sémantique de DyALog séquentiel.

ATTENTION: si les processeurs envoient les objets créés au serveur au fur et à mesure de leur création, alors: pour O_a compatible avec O_e, on voit dans le cas suivant que O_e ne s'appliquera jamais à O_a et vice-versa. Pourtant cette application est indispensable au bon déroulement de DyALog.

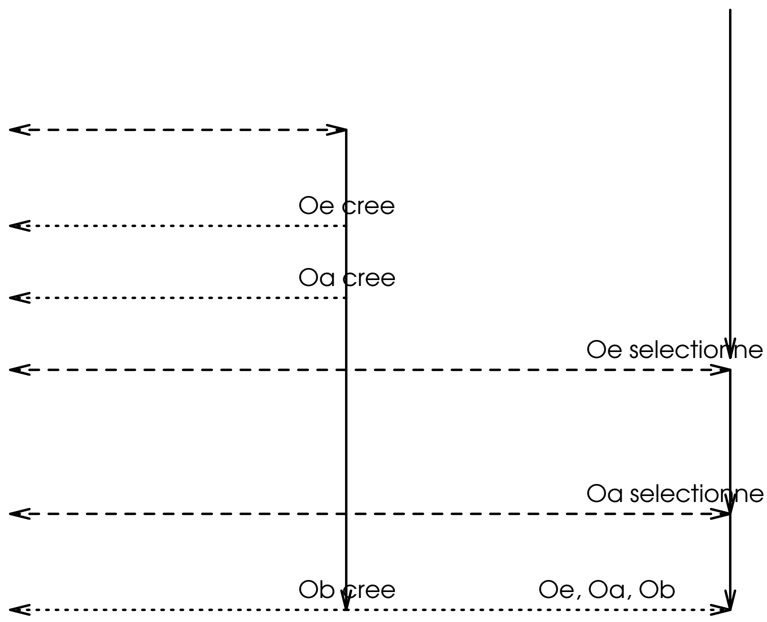


FIG. 3.12 - *Sémantique de I^2.*

Deux propositions de correction:

1. Après la création d'un objet, le processeur le communique simultanément au serveur et aux $(p-2)$ autres processeurs: ce modèle est sensiblement le même que I': on ne gagne ici de temps qu'en évitant les synchronisations au moment de l'envoi des tâches à effectuer.

2. Le Maître peut sélectionner un objet dans l'agenda si et seulement si celui-ci a été communiqué à tous les processeurs: cette idée est certainement la meilleure des deux proposées. De la même façon que pour I, il faut tenir à jour un tableau de listes des objets sélectionnés et non encore communiqués aux processeurs. Ceci afin de savoir à tout instant si le processeur auquel on affecte une tâche connaît bien l'ensemble des tâches qui ont déjà été sélectionnées.

3.7.3 Complexité.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant [12]:

tâche	coût
1	$m((p-1)x' + C + g)$
2	$m(p-1)x'I_t$
3	$m(R)$
4	$m(xA)$
5	$m(xS)$
6	$m((p-2)x'g) + l$

Avec
x : nombre d'objets O_2 applicables à un O_1.

12. les notations sont définies page 11

x': *nombre de* O_3 *créés (après test de subsomption) pour un* O_1.

Cette boucle sera répétée $\frac{n}{m(p-1)}$ fois au cours de l'exécution.
Ce qui nous donne la complexité globale $\mathbf{C_{I^2}}$ suivante:

$\mathbf{C_{I^2}} = \frac{n}{m(p-1)}(m[C + R + x(A + S) + x'(p - 2)g + (p - 1)x' + g + (p - 1)x'I_t] + l)$

3.7.4 Saturation du serveur.

Un tour de boucle esclave coûte:$m[C+R+x(A+S)+x'(p-2)g+(p-1)x'+g+(p-1)x'I_t]+l$

Une communication au serveur coûte:$mx'g + l + mx'I_a + C + (1 + mp)g$

Pour que le serveur n'atteigne pas son niveau de saturation, il faut:

$$1 \text{ tour de boucle esclave} \geq (p-1)\text{communications au serveur} \tag{3.18}$$

Les valeurs acceptables de p (pour lesquelles le serveur ne saturera pas) sont comprises entre 1 et la racine positive de l'inéquation:

$-p^2[mg] + p[m(x'(1 + I_t - I_a) + g) - C - l - g] + [m(C + R + x(A + S) + x'(I_a - I_t) - 2x'g - x + g) + 2l + C + g] \geq 0$

3.7.5 Remarques.

Les côtés positifs.

- Le fait de grouper les tâches a des effets bénéfiques quant au gain de temps:
  - La variance entre les temps de réalisation des groupes de tâches est inférieure à celle de réalisation des tâches.
  - Le nombre de communications est diminué.
  - Le nombre des synchronisations est diminué.

Les côtés négatifs.

- La trace d'exécution d'un tel algorithme ne sera certainement pas facile à suivre à la main du fait de l'inexistence de synchronisation globale des processeurs.

Implantation. Il ne faut pas oublier de tenir à jour la liste des objets qui ont été sélectionnés dans l'agenda.

3.8 La solution S: évolution du modèle Maître/Esclave.

L'idée fondatrice de cet algorithme est l'utilisation du **modèle Maître/Esclave, sans processeur serveur**. Le but premier de cette nouvelle idée est **l'économie d'un processeur**: le serveur. Il est alors libre et peut exécuter des tâches de recherche, application et test de subsomption, tout comme les autres processeurs.

La programmation de cet algorithme demande la mise au point d'un protocole de sélection des objets dans l'agenda: il faut éviter que deux processeurs sélectionnent et traitent le même objet. Ce cas est le plus vraisemblable lorsque deux processeurs sélectionnent simultanément une nouvelle tâche.

Voici un protocole pouvant résoudre ce problème:

1. **au top** 0:
   - envoi des objets créés par i
   - sélection du prochain objet dans l'agenda
   - mise en route du chrono
2. **au top** g:
   - si aucun message n'arrive: alors continuer
   - si expéditeur > i alors
     - i prend le premier élément de l'agenda
     - expéditeur prend le second
   - sinon
     - expéditeur prend le premier élément de l'agenda
     - i prend le second

L'algorithme est décrit par:

Par $i = 1$ à p **faire**

  Tant que non Fini **faire**
    [1] insérer les objets de la file F_i dans la table des objets
      vider la file F_i
    [2] sélectionner une tâche dans l'agenda
      $O_{1_i} \leftarrow$ objet sélectionné
    [3] recherche d'unification sur O_{1_i} dans toute la table des objets
    [4] application des O_{2_i} aux O_{1_i} pour donner les O_{3_i}
    [5] test de subsomption faible sur les O_{3_i}
      $O_{3_i} \leftarrow$ objets survivants
      enfiler O_{3_i} dans F_i
    [6] envoyer O_{3_i} aux $(p-1)$ autres processeurs même si l'ensemble est vide
      enfiler les objets reçus au cours du traitement dans F_i
      dépiler de l'agenda autant d'objets que de processeurs
      ont envoyé des objets, moins 1.

    si Agenda_Vide et Aucune_Tâche_En_Cours **alors** Fini $\leftarrow$ Vrai

    fin_tant_que

  fin_par

3.8.1 Une idée de l'exécution de S.

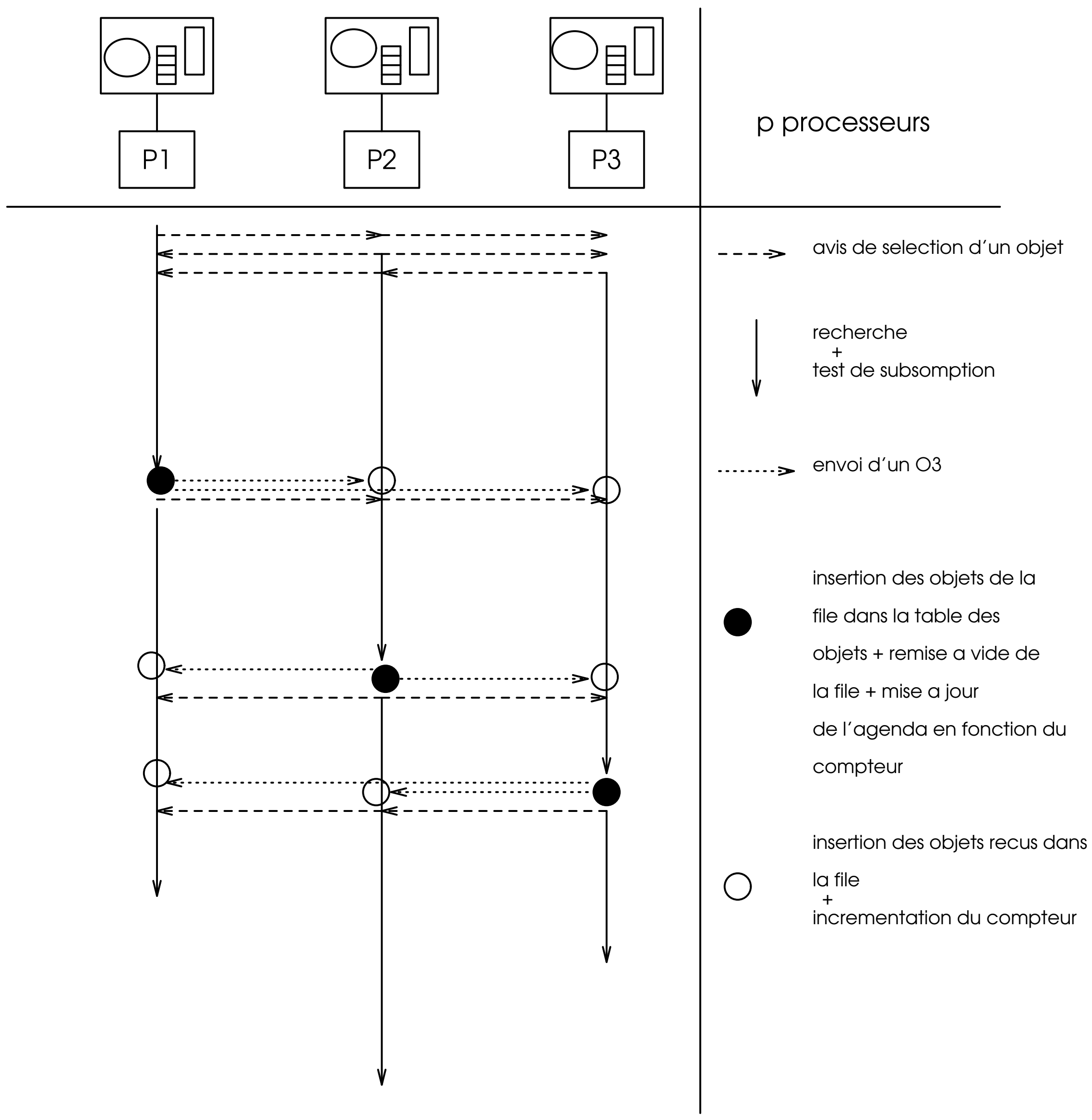


FIG. 3.13 - *Solution S: exécution.*

3.8.2 Conservation de la sémantique.

La preuve de la sémantique de S est la même que celle de I' puisque S n'est qu'une adaptation du modèle Maitre/Esclave. La sémantique de S est donc fidèle à celle de DyALog séquentiel.

3.8.3 Complexité.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant [13]:

tâche	coût
1	$x'pI_t$
2	$g+C$
3	R
4	xA
5	xS
6	$px'g+l$

Avec
x : nombre d'objets O_2 applicables à un O_1.
x': nombre de O_3 créés (après test de subsomption) pour un O_1.

Cette boucle sera répétée $\frac{n}{p}$ fois au cours de l'exécution.
Ce qui nous donne la complexité globale $\mathbf{C_S}$ suivante:

$$\mathbf{C_S} = \frac{n}{p}[R + x(A+S) + x'p(I_t+g) + g + l + C]$$

3.8.4 Remarques.

Les côtés positifs.

- On ne rencontre pas de problème de saturation du serveur.
- Il est évident de tenir à jour la liste des objets déja sélectionnés.
- Cet algorithme est utilisable dans le cas d'une machine disposant d'un petit nombre de processeurs.

3.9 Une solution hybride née de G et de I.

Là encore, la machine utilisée est une machine à mémoire distribuée. Chaque processeur dispose d'une mémoire où sont stockés la table des objets, l'agenda et le lot de piles utilisées par le processeur.

L'idée de base est **une recherche séparée dans la table des objets, secondée par une recherche partagée** afin d'aider un processeur pour lequel la recherche s'avérerait plus longue que pour les autres. Ainsi il ne resterait pas de processeur inactif qui attendrait la synchronisation globale.
C'est à dire qu'une fois qu'un processeur aurait fini sa tâche, il viendrait en aide à un autre qui n'aurait pas encore fini.

L'algorithme s'écrit:

13. les noatations sont définies page 11

Tant que Agenda_Non_Vide **faire**

Par $i = 1$ à p **faire**

[1] $O_1 \leftarrow i^{ème}$ objet de l'agenda

dépiler p objets de l' agenda

[2] recherche d'unification dans toute la table des objets

[3] application des O_2 trouvés à O_1 pour donner les O_3

[4] test de subsomption faible sur les O_3 dans toute la table

[5] si Pas_tous_fini alors **aider_le_plus_proche**

[6] envoi des objets O_3 créés aux $(p-1)$ autres processeurs

fin_par

Par $i = 1$ à p **faire**

[7] insertion des objets reçus dans la table des objets

[7] insertion des objets reçus dans l'agenda

fin_par

fin_tant_que

avec

aider_le_plus_proche = vider_piles_locales
charger_piles_processeur_aidé
partager_table
recherche_avec_algo_GI

3.9.1 Une idée de l'exécution.

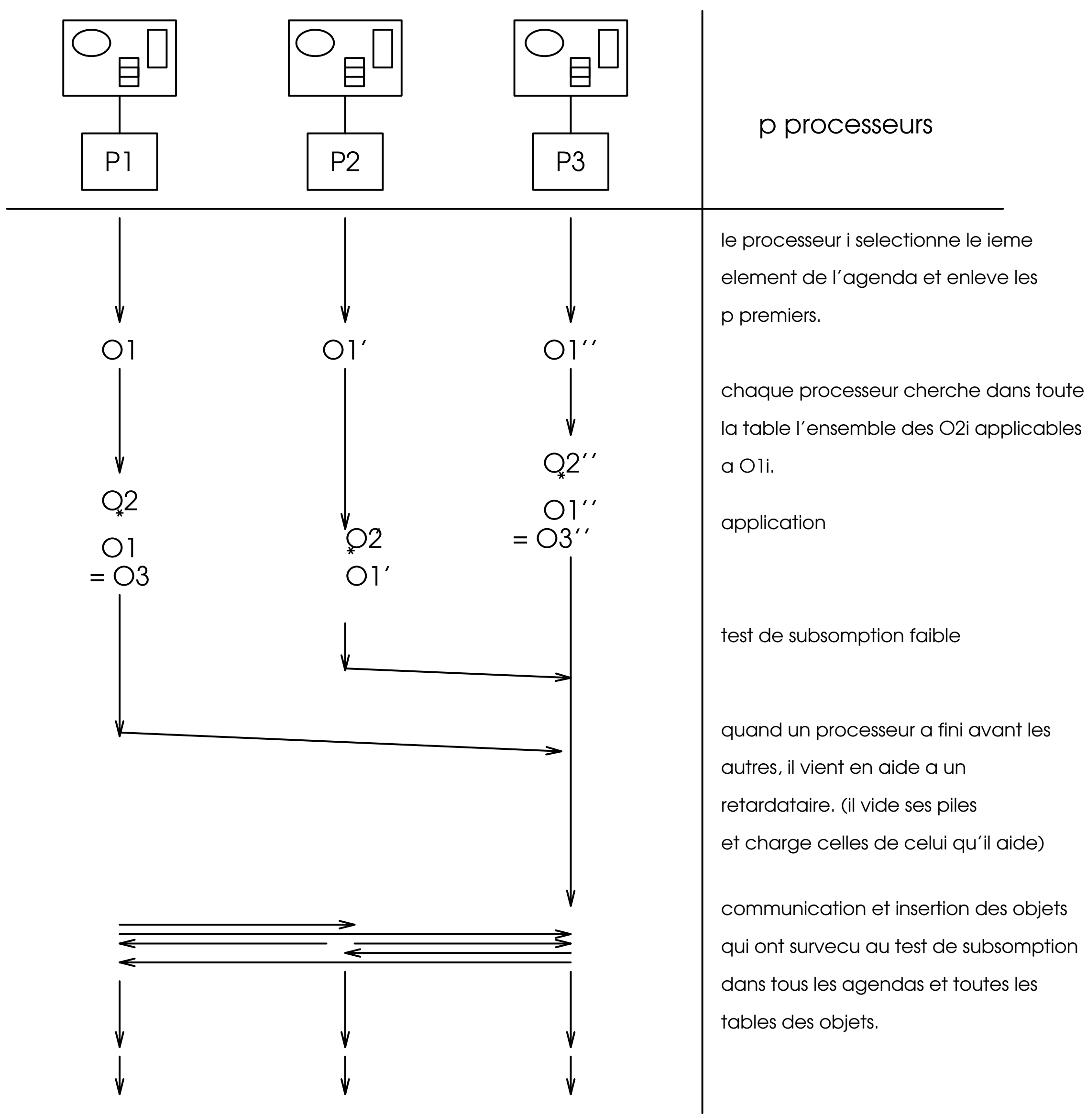


FIG. 3.14 - *Solution hybride: exécution.*

3.9.2 Conservation de la sémantique.

GI est construit sur les algorithmes G et I. Or chacun d'eux a la même sémantique que DyALog séquentiel. Il est donc trivial que GI conserve lui aussi la sémantique de DyALog séquentiel.

3.9.3 Complexité.

Les coûts de chaque étape d'un tour de boucle de cet algorithme sont récapitulés dans le tableau suivant [14]:

14. les notations sont définies page 11

tâche	coût
1	C
2	R
3	xA
4	xS
5	$aide + l$
6	$x'(p-1)g + l$
7	$x'(I_t + I_a)$

Avec
x *: nombre maxi d'objets* O_2 *trouvés par un même processeur.*
x'*: nombre total sur l'ensemble des processeurs des objets qui ont survécu au test de subsomption.*

Cette boucle sera répétée $\frac{n}{p}$ fois au cours de l'exécution.
Ce qui nous donne la complexité globale $\mathbf{C_{GI}}$ suivante:

$$\mathbf{C_{GI}} = \frac{n}{p}(C + R + x(A+S) + aide + 2l + x'(p-1)g + x'(I_t + I_a))$$

Nota: *aide* est une valeur qu'il ne nous est pas possible d'estimer. En effet, on ne peut pas, à l'heure actuelle, mesurer les écarts entre des temps d'exécution de deux tours de boucle principale de DyALog séquentiel. Cet algorithme restera donc au stade d'idée.

3.9.4 Remarques.

Les problèmes qui se posent:

- Comment évaluer le gain/la perte de temps occasionné par l'aide d'un processeur envers un autre?
- Il faut conserver l'identité du processeur qui est originellement en charge de la tâche à exécuter, pour que lui seul l'envoie aux autres processeurs (un envoi en double serait source de perte de temps).

Les problèmes qui ne se posent pas:

- Un processeur qui a fini sa tâche peut, pour aller en aider un autre, vider sa pile des opérations courantes (ne doit-elle d'ailleurs pas être vide pour une exécution terminée?) et charger la pile courante de celui qu'il aide.

Les côtés positifs.

- Un algorithme rythmé par un chef d'orchestre: la synchronisation globale.
- Pas de perte de temps dans l'attente de la synchronisation globale.
- Une trace d'exécution facile à suivre.

Les côtés négatifs.

- Il faut implanter une notion supplémentaire: le choix du processeur aidé.
- Pas facile, à l'heure actuelle de savoir si cette idée est intéressante à implanter.

3.10 Tableau récapitulatif des propriétés des algorithmes:

		Mémoire		Propriétés						
Nom	Pge	Tab Obj	Agen	Serv	Rech part	Rech seul	Subθ part	Subθ seul	Gram det	Gram ambi
G1	13	X	X		X			X	X	X
G21	16	X	X		X		X		X	X
G22	16	X	X		X		X		X	X
I	26	X	X			X		X		X
M/E	29	$P_{i\neq 0}$	P_0	X		X		X		X
I^2	34	$P_{i\neq 0}$	P_0	X		X		X		X
S	38	X	X			X			X	
GI	41	X	X		X	X		X		X

Légende:
Pge: page de présentation de l'algorithme.
Tab. Obj.: Table des objets présente dans chaque mémoire.
Agen.: Agenda présent dans chaque mémoire.
Serv: Utilisation d'un serveur dans l'algorithme.
Rech. part.: l'algorithme utilise une recherche partagée.
Rech. seul: l'algorithme utilise une recherche exécutée par un processeur seul.
Subθ part: l'algorithme utilise un test de subsomption partagé.
Subθ seul: l'algorithme utilise un test de subsomption exécuté par un processeur seul.
Gram. det.: algorithme adapté aux grammaires déterministes.
Gram. ambi.: algorithme adapté aux grammaires ambiguës.

3.11 Une intuition.

On peut, à l'heure actuelle penser qu'il n'existera peut être pas de solution optimale vraiment unique d'algorithmique parallèle pour DyALog. En effet, il est possible que le meilleur algorithme ne soit pas toujours le même en fonction du problème posé (grammaire de type Langage Naturel, faiblement ambiguë ou déterministe). Ce résultat pourra aussi être dépendant des paramètres de la machine sur laquelle l'exécution aura lieu.

On peut aussi imaginer qu'un nouvel algorithme hybride naisse: un algorithme qui s'inspirerait des solutions montrées optimales pour certains problèmes.

Chapitre 4

Les idées rejetées.

Nous allons, dans ce chapitre recenser les idées que nous avons eues et qui se sont avérées décevantes.

4.1 La répartition des données.

Une première idée de parallélisation fut d'utiliser le modèle BSP. C'est à dire un ensemble de processeurs munis chacun de leur mémoire, qui peuvent échanger des informations via un réseau d'interconnection.

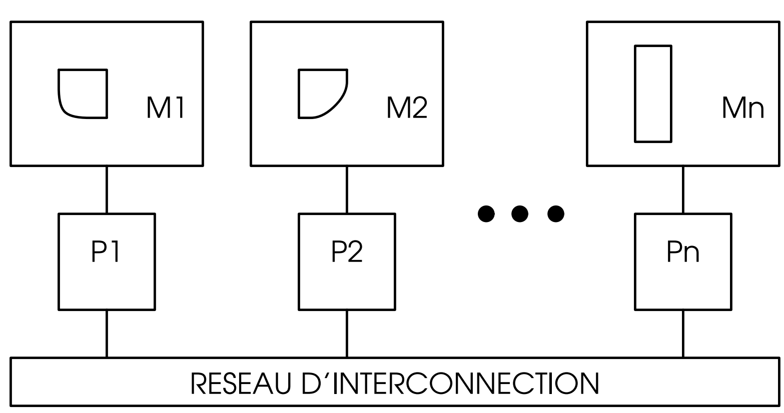


FIG. 4.1 - *Modèle de répartition des données.*

On pensait **répartir l'ensemble des données** (agenda, table des objets...) de DyALog sur les mémoires associées aux différents processeurs. La table des objets pouvait être divisée en blocs répartis sur les mémoires des $n-1$ premiers processeurs, tandis que l'agenda était présent sur la n-ième mémoire.

Mais une telle répartition s'est vite montrée irréaliste pour deux raisons. Tout d'abord, la structure partagée utilisée dans la structure de données de DyALog: ses éléments sont trop étroitement liés pour être séparés tout en conservant leur efficacité (optimisation de la place mémoire et du temps d'accès). Ensuite, on ne peut pas garantir la conservation d'une répartition équilibrée des charges entre les mémoires des processeurs. En considérant qu'un processeur traite les éléments présents dans la partie de table des objets présents dans sa mémoire, un déséquilibre important peut conduire à une efficacité très médiocre. Dans le cas d'échanges de données entre les processeurs, ce sont les communications qui deviennent très coûteuses en temps.

4.2 La répartition des tâches sur une mémoire partagée.

Une autre idée de mise en place de parallélisation pour DyALog fut l'utilisation d'une mémoire partagée (qui résolvait le problème rencontré ci-dessus) à laquelle accédait un ensemble de processeurs.

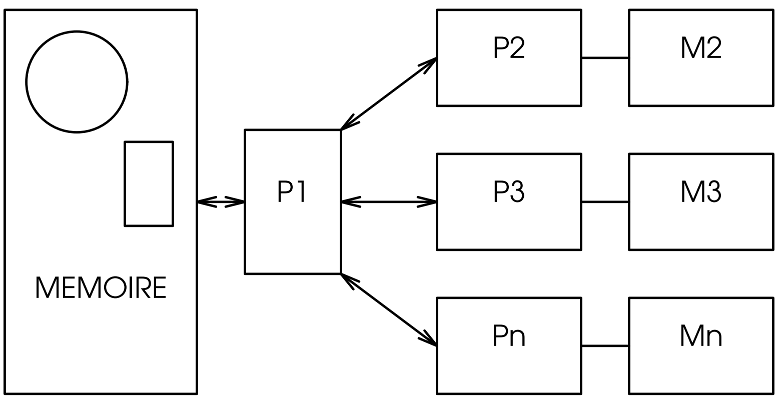


FIG. 4.2 - *Modèle de répartition des tâches.*

Comme l'accès à la mémoire est limité à un processeur à la fois, un des processeurs était utilisé comme serveur et communiquait le contenu de la mémoire demandé par les autres.

Mais ce modèle n'était pas valable dans le cas d'un nombre important de processeurs: une longue file d'attente se serait vite formée pour entrer en communication avec le processeur serveur P1, ce qui est assimilable à un goulet d'étranglement: en effet la taille et le nombre des informations à communiquer auraient été trop importants. Ceci aurait pu nous conduire à une complexité aussi importante que celle de DyALog séquentiel, ou pire encore.

4.3 Le découpage dynamique de la table des objets.

Les solutions G1 et G2 présentées dans le chapitre précédent aux pages 13 et 16 sont construites sur une même idée : chaque processeur dispose de toute la table des objets dans sa mémoire, pourtant il n'est chargé d'en explorer qu'une partie. Cette répartition des "portions" de table des objets à traiter entre les processeurs se fait dynamiquement.

Un problème reste cependant à résoudre: comment faire une répartition dynamique des objets de la table, la plus équitable possible (en place et en temps de traitement), dans laquelle tous les processeurs auraient le même nombre de tests à effectuer?

Plusieurs idées se présentent:

- **une répartition fixe** qui ne dépend pas de la répartition des objets dans la table. Mais il y a bien peu de chances que ce soit efficace: on ne peut en aucun cas prédire quelle partie de la table va grandir le plus vite.

– **l'utilisation d'un compteur d'objets**. On ajoute un compteur à chaque nœud de la table. Il contient le nombre d'objets présents dans la structure, en dessous de lui. Le nœud principal de la structure contient donc le nombre total des objets contenus dans la table: chaque processeur i est chargé de l'étude de la i ème tranche de $\frac{n}{p}$ objets. On pourrait penser, à première vue que cette solution permet une répartition parfaite des recherches à effectuer entre les processeurs; mais elle n'aboutit pas au partage optimal.

On donne ici une intuition de la raison pour laquelle la partition obtenue n'est pas nécessairement équitable. On peut représenter la structure de la table des objets comme suit:

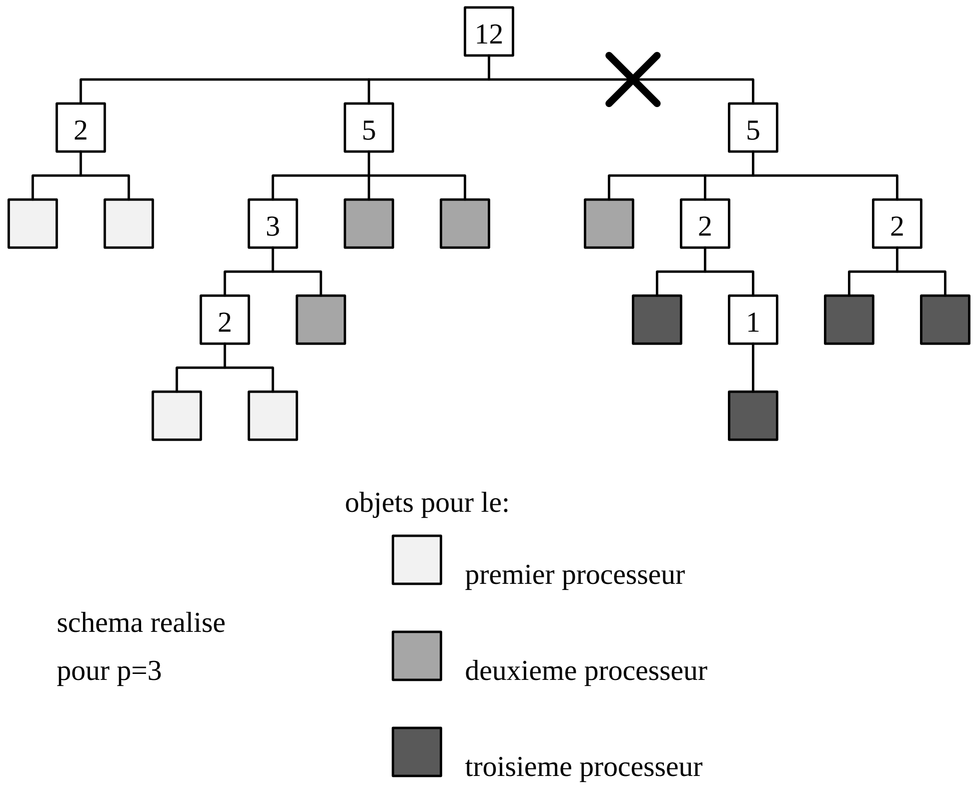


FIG. 4.3 - *Modèle de répartition dynamique de la table des objets: les compteurs.*

Si dans cet exemple, dès le premier test d'unification le chemin le plus à droite s'avère inexploitable (unification impossible), alors toute l'arborescence qui découle de cette liaison n'a plus besoin d'être parcourue: on ne pourra y trouver aucun terme unifiable, et ce par définition même de l'unification. Ainsi, le troisième processeur n'a plus de travail à effectuer alors que le premier processeur doit traiter (au plus) 4 objets et le second 3 (au plus).

Cette procédure ne garantit donc pas une répartition optimale des objets de la table entre les processeurs.

On vient de montrer que la partition qui consiste à affecter $\frac{n}{p}$ objets à chaque processeur n'était pas optimale. Pourtant, nous n'avons, à ce jour, pas trouvé de répartition plus efficace. En effet on sent bien qu'elle n'est pas non plus mauvaise. On peut dire que le processeur i devra, dans le pire cas traiter $\frac{n}{p}$ objets.

Chapitre 5

Essais comparatifs.

Après les phases d'étude des faiblesses de DyALog et de mise au point des algorithmes parallèles, vient le moment d'évaluer les performances des algorithmes proposés. Ensuite pourra-t-on dire si la parallélisation de DyAlog peut procurer un gain de temps, ou non. En effet, à ce stade de l'étude, après analyse des temps calculés, on pourra tirer une des trois conclusions suivantes:

- les résultats attendus en parallèle sont meilleurs que ceux de DyALog séquentiel $\Rightarrow$ DyALog est parallélisable.
- les résulats attendus ne sont pas toujours meilleurs que ceux de DyALog séquentiel, mais le sont parfois $\Rightarrow$ la parallélisation est à envisager sur certaines machines, pour certains types de grammaires.
- les résultats sont toujours pires que ceux de la version séquentielle $\Rightarrow$ DyALog n'est pas parallélisable.

5.1 Les données utilisées.

Nous avons repris les mesures faites sur DyALog séquentiel, et évalué théoriquement les performances que devraient accomplir les algorithmes parallèles.

Nous avons conservé nos trois types de grammaires et avons réalisé les calculs sur plusieurs grammaires de chaque type afin d'avoir un échantillon plus représentatif de la réalité.

Il est proposé, dans les pages suivantes: le calcul du temps d'exécution escompté, pour chaque grammaire, pour un échantillon de 76 machines[1], pour chaque algorithme parallèle proposé.

5.2 Le mode de calcul.

Afin de réaliser les estimations de coûts de chaque algorithme pour un ensemble de machines sur un échantillon de 3 types de grammaires, nous nous sommes efforcés d'utiliser au mieux les mesures réalisées sur DyALog séquentiel.

1. les paramètres BSP de ces machines proviennent de http://www.BSP-Worldwide.org/implmnts/oxtool/ , page la plus complète disponible à ce jour

C'est en observant des correspondances entre les mesures faites sur DyALog et des fragments d'expressions de complexités des algorithmes, que nous sommes arrivés à des résultats proches de la réalité.

En effet, on constate que:

$$\begin{aligned} nC &= C_{seq} \\ nR &= R_{seq} \\ nxA &= A_{seq} \\ nxS &= S_{seq} \\ nx'I_t &= I_{t_{seq}} \\ nx'I_a &= I_{a_{seq}} \end{aligned}$$

où $C_{seq}, R_{seq}, A_{seq}, S_{seq}, I_{t_{seq}}, I_{a_{seq}}$ sont les temps totaux (ou séquentiels) dans les fonctions C correspondantes.

et

$$\begin{aligned} nx' &= \text{nombre d'insertions dans agenda} \\ &= \text{nombre de tours de boucles} \\ &= n \\ \\ nx &= \text{nombre d'applications en séquentiel} \\ &= n_{appli} \end{aligned}$$

Avec de telles équivalences, on réécrit les complexités des algorithmes parallèles en fonction de valeurs mesurables (et déjà mesurées) sur DyALog séquentiel.

5.2.1 Algorithme G1.

$$\begin{aligned} C_{G1} &= n(C + \frac{R}{p} + \frac{x}{p}(A+S) + x'(p-1)g + l + x'(I_t + I_a)) \\ &= nC + n\frac{R}{p} + n\frac{x}{p}(A+S) + nx'(p-1)g + nl + nx'(I_t + I_a) \\ &= C_{seq} + \frac{R_{seq}}{p} + \frac{A_{seq}}{p} + \frac{S_{seq}}{p} + n(p-1)g + nl + I_{t_{seq}} + I_{a_{seq}} \end{aligned}$$

5.2.2 Algorithme G21.

$$C_{G21} = n(C + \frac{R}{p} + \frac{x}{p}A + (p-1)g(\frac{x}{p} + x') + 2l + \frac{x}{p}S + \Pi + x'(I_t + I_a) + \varepsilon)$$

$$
\begin{aligned}
&= nC + n\frac{R}{p} + n\frac{x}{p}A + n(p-1)g(\frac{x}{p}x + x') + 2nl + n\frac{x}{p}S + n\Pi + nx'(I_t + I_a) + n\varepsilon \\
&= C_{seq} + \frac{R_{seq}}{p} + \frac{A_{seq}}{p} + (p-1)g(\frac{n_{appli}}{p} + n) + 2nl + \frac{S_{seq}}{p} + \\
&\quad + \log_2 p(ng + nl + n) + I_{t_{seq}} + I_{a_{seq}}
\end{aligned}
$$

5.2.3 Algorithme G22.

$$
\begin{aligned}
C_{G22} &= n(C + \frac{R}{p} + \frac{x}{p}A + (p-1)(g\frac{x}{p} + gx' + l) + \frac{x}{p}S + l + x'(I_t + I_a) + \varepsilon') \\
&= nC + n\frac{R}{p} + n\frac{x}{p}A + n(p-1)(g\frac{x}{p} + gx' + l) + n\frac{x}{p}S + nl + nx'(I_t + I_a) + n\varepsilon \\
&= C_{seq} + \frac{R_{seq}}{p} + \frac{A_{seq}}{p} + (p-1)(nl + g(\frac{n_{appli}}{p} + n)) + \frac{S_{seq}}{p} + nl + I_{t_{seq}} + I_{a_{seq}}
\end{aligned}
$$

5.2.4 Algorithme I.

$$
\begin{aligned}
C_I &= \frac{n}{p}(C + R + x(A+S) + x'(p-1)g + l + px'(I_t + I_a)) \\
&= n\frac{C}{p} + n\frac{R}{p} + \frac{n}{p}x(A+S) + \frac{n}{p}x'(p-1)g + \frac{n}{p}l + nx'(I_t + I_a) \\
&= \frac{C_{seq}}{p} + \frac{R_{seq}}{p} + \frac{A_{seq}}{p} + \frac{S_{seq}}{p} + \frac{n}{p}(p-1)g + \frac{n}{p}l + I_{t_{seq}} + I_{a_{seq}}
\end{aligned}
$$

5.2.5 Algorithme I'.

$$
\begin{aligned}
C_{I'} &= \frac{n}{p-1}[(p-1)x'I_t + C + g + R + x(A+S) + x'(p-1)g + l] \\
&= nx'I_t + \frac{n}{p-1}C + \frac{n}{p-1}g + \frac{n}{p-1}R + \frac{n}{p-1}x(A+S) + nx'g + \frac{n}{p-1}l \\
&= I_{t_{seq}} + \frac{C_{seq}}{p-1} + \frac{n}{p-1}g + \frac{R_{seq}}{p-1} + \frac{A_{seq}}{p-1} + \frac{S_{seq}}{p-1} + ng + \frac{n}{p-1}l
\end{aligned}
$$

5.2.6 Algorithme I^2.

$$
\begin{aligned}
C_{I^2} &= \frac{n}{m(p-1)}[m(C + R + x(A+S) + (p-2)x'g + (p-1)x' + g + \\
&\quad +(p-1)x'I_t) + l] \\
&= \frac{n}{p-1}(C + R + x(A+S)) + n\frac{(p-2)}{(p-1)}x'g + nx' + \frac{n}{p-1}g + \\
&\quad +nx'I_t + \frac{n}{m(p-1)}l
\end{aligned}
$$

$$= \frac{C_{seq}}{p-1} + \frac{R_{seq}}{p-1} + \frac{A_{seq}}{p-1} + \frac{S_{seq}}{p-1} + n\frac{(p-2)}{(p-1)}g + n + \frac{n}{p-1}g + \\ +I_{t_{seq}} + \frac{n}{m(p-1)}l$$

5.2.7 Algorithme S.

$$\begin{aligned} C_{I^2} &= \frac{n}{p}[R + x(A+S) + px'(I_t + g) + g + l + C] \\ &= \frac{n}{p}R + \frac{n}{p}x(A+S) + nx'I_t + nx'g + \frac{n}{p}g + \frac{n}{p}l + \frac{n}{p}C \\ &= \frac{R_{seq}}{p} + \frac{A_{seq}}{p} + \frac{S_{seq}}{p} + \frac{C_{seq}}{p} + I_{t_{seq}} + \frac{n}{p}g + \frac{n}{p}l + ng \end{aligned}$$

Remarque On ne devra pas oublier, au moment des calculs, d'exprimer le paramètre BSP g dans l'unité qui convient. En effet, le constructeur donne g en "flops/mot", alors que nos formules utilisent g en "flops/objet". Il faudra donc multiplier le g du constructeur par la taille moyenne d'un objet. Celle-ci est calculée à partir du nombre de liaisons créées au cours de l'exécution séquentielle, et du nombre d'objets intervenant dans cette même exécution.

On suppose... Dans les calculs qui sont présentés dans la suite, nous avons supposé que tout processeur était occupé à tout moment. C'est à dire que nous avons négligé les attentes avant synchronisation. Nous n'avons pas non plus tenu compte du nombre d'objets présents dans l'agenda: ceci peut pourtant être une barrière au bon fonctionnement de certains algorithmes dans le cas de grammaires pas assez ambiguës.

5.3 Les mesures et les résultats des calculs.

Des résultats de mesures complets vous sont proposés en Annexe[2].

5.4 Les leçons à tirer.

5.4.1 Pour les grammaires de type Langage Naturel.

Dans tous les cas (pour une grammaire donnée, sur un ensemble de machines), il existe au moins un algorithme parallèle bien meilleur que l'algorithme séquentiel. Cet algorithme obtient en général de bon résultats sur plusieurs machines.

Pourtant il faut rester vigilant: ce n'est pas toujours sur la machine qui compte le plus grand nombre de processeurs que l'on obtient les meilleurs performances.
Il existe aussi des machines sur lesquelles tous les résultats obtenus sont pires que ceux de la version séquentielle.

2. ou sont disponibles sur simple demande à debourge@lifo.univ-orleans.fr ou Eric.Clergerie@inria.fr

Les algorithmes de la famille I (c'est à dire I, I', I^2, S) remportent souvent le meilleur score quant au temps d'exécution. Il arrive que l'algorithme G1 atteigne aussi cette performance.

Pour les grammaires de type Langage Naturel, les meilleurs scores sont (presque) toujours obtenus sur les mêmes machines avec les mêmes algorithmes. Il en est de même pour les résultats catastrophiques: les résultas plusieurs fois pires que ceux obtenus par DyALog séquentiel.

Les facteurs d'accélération sont importants et peuvent dépasser 20.

On a, par exemple, sur la grammaire Bio_123, les résultats suivants:

Pour CRAY T3D [3]:

p	l	g	G1	G21	G22	I	I'	I^2	S
1	68	0.3	124	124	124	124			124
2	164	1	66	67	66	64	125	124	64
4	168	0.8	36	38	38	34	44	44	34
8	175	0.8	21	25	27	19	21	21	19
9	383	1.2	21	29	35	17	19	18	17
16	181	1	14	19	27	11	11	11	11
25	486	1.5	13	27	69	8	9	8	8
32	201	1.4	11	17	41	7	7	7	7
64	148	1.7	10	16	55	5	5	5	5
128	301	1.8	13	25	196	4	4	4	4
256	387	2.4	22	40	494	4	4	4	4

On voit que les meilleurs résultats sont obtenus par I, I', I^2, S pour les machines à 64 et 128 processeurs. L'accélération peut atteindre 34,25 avec S sur 256 processeurs.

Par contre, les CRAY T3D à 128 et 256 processeurs demandent plus de temps pour G22 que pour la version séquentielle.

Le Pentium PRO, sur la grammaire Tag_3000 nous donne:

p	l	g	G1	G21	G22	I	I'	I^2	S
1	85	1	3	3	3	3			3
2	52745	484.5	16	44	30	9	18	5	10
4	139981	1128.5	40	152	151	11	15	4	12
8	539159	1994.1	150	720	1141	20	23	5	20
10	826054	2436.3	230	1172	2181	24	27	5	25
16	288444373	3614.6	75692	454013	1210628	4732	5048	508	4732

Une telle machine ne permet pas d'obtenir de meilleurs résultats que ceux de DyALog séquentiel (l'exécution de DyALog séquentiel sur la même machine pour la même grammaire durerait 3 secondes).

3. pour tous les résultats: l est en flops, g en flops/word et les temps d'exécution des algorithmes sont en secondes.

5.4.2 Pour les grammaires faiblement ambiguës.

Dans cette famille de grammaires, on n'obtient presque pas de résultats pires qu'en séquentiel. Pourtant les algorithmes les plus rapides ne sont pas les mêmes que ceux de la famille Langage Naturel: G1, G21 et G22 ont rejoint les algorithmes I, I', I^2, S qui etaient deja performants pour les grammaires LN.
Souvent, pour un nombre important de processeurs, tous les algorithmes parallèles atteignent le meilleur score.

Exemple, avec Hanoi_18 sur Origin 2000:

p	l	g	G1	G21	G22	I	I'	I^2	S
1	286	1.36	5	5	5	5			5
2	804	8.26	3	3	3	3	5	5	3
3	1313	8.36	2	2	2	2	3	3	2
4	1789	10.24	1	1	1	1	2	2	1
5	2474	11.06	1	1	1	1	1	1	1
6	2963	12.25	1	1	1	1	1	1	1
7	3867	14.28	1	1	1	1	1	1	1

Ici, il est très clair que les algorithmes atteignent le meilleur score dès que $p = 5$.

Mais il faut ajouter que l'accélération est limitée par le faible temps de calcul séquentiel dû à la taille de la grammaire.

5.4.3 Pour les grammaires déterministes.

Pour ce type de grammaires, les résultats des algorithmes I, I', I^2, S ne sont pas considérés comme recevables. En effet on sait que le cardinal de l'agenda reste égal à un tout au long de l'exécution, pour ce type de grammaire. Les résultats des algorithmes de la famille I sont donc faussés. On sent bien que le résulat réel serait catastrophique.

C'est l'algorithme G1 qui est souvent le plus compétitif.
Il arrive, là encore, de rencontrer des résultats pires en parallèle qu'en séquentiel.

5.5 Généralités.

Les résultats obtenus ne sont bien sûr que des approximations.
Pourtant, on peut penser qu'elles sont assez proches de la réalité si l'on compare les résultats attendus sur une Sparc Station 5 à 1 processeur avec les résultats obtenus en séquentiel avec le même type de machine.

Chapitre 6

Le détail des implantations.

Nous avons décrit, dans les chapitres précédents, les idées directrices des algorithmes parallèles proposés pour DyALog. Le présent chapitre présente des détails nécessaires à une implantation parallèle de DyALog:

- le stockage des informations.
- le transfert des informations.

Ils sont valables pour tous les algorithmes qui ont été proposés dans le chapitre 5.

6.1 Le stockage des informations.

6.1.1 Zones globales

- **Agenda**: Liste ordonnée des tâches à traiter. Une tâche est représentée par un objet. Cette liste est extensible.
- **Folcmb Table**: Table des constantes.
- **Heap**: Zone de stockage des termes et des objets. Cette zone est allouée au fur et à mesure.
- **Object Table**: (Table des objets). Elle regroupe l'ensemble des objets construits et tabulés par DyALog. Table extensible et indexée par arborescence.
- **Stream Table**: Tableau non extensible des stream ouverts (pour les entrées/sorties). (Table de correspondances).
- **Symbol Table**: Tableau non extensible des symboles.

6.1.2 Zones locales

- **Heap Buffer**: Zone de création temporaire des termes logiques avant de les tester par rapport à ceux donnés par la Term Table.
- **Layer Stack**: Pile de chargement des fiches pour le partage de structure.
- **Registers**: Tableau non extensible de registres systèmes et registres utilisateurs pour DyA-Log.

– **Trail Stack**: Pile de sauvegarde d'informations temporaires (liaisons d'unification et de subsomption...).

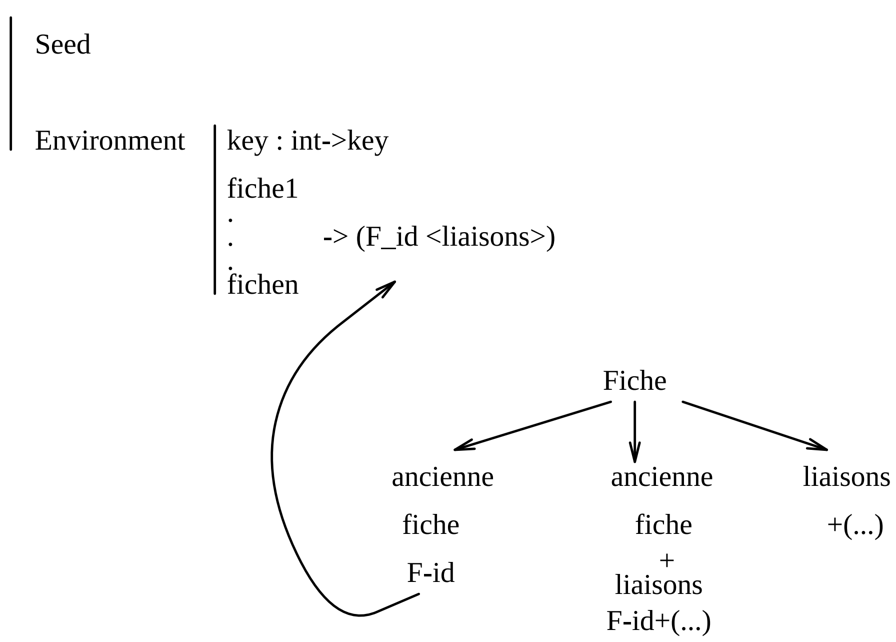
– **Variable Table**: Tableau non extensible fournissant une entrée vers les liaisons d'unification et de subsomption pour chaque variable. L'entrée n est associée à la variable d'indice n.

6.2 Le transfert des informations.

Pour un processeur i qui envoie un terme ou un objet O à un processeur j, le principe retenu[1] pour le transfert des objets est le suivant:

6.2.1 Codage d'un objet.

Un objet a la forme suivante:



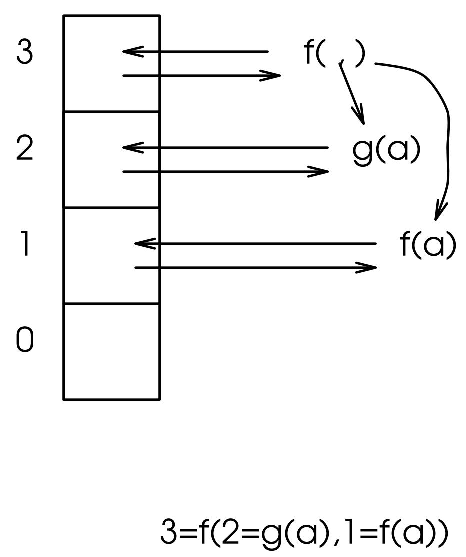
où une liaison s'écrira:

liaison ::=	<var>	<term>	<shift>
	$\downarrow$	$\downarrow$	$\downarrow$
	V_id	T_id	shift

Pour les termes, DyALog séquentiel utilise des pointeurs.
Pour la version parallèle, on utilisera des handles[2]: pointeurs à deux directions.

Par exemple, pour f(g(a),f(a)), on pourra avoir:

1. mis au point conjointement avec E. Villemonte de la Clergerie
2. poignées



Il faudra donc, pour le transfert d'un objet:

- 1 id pour "seed"
- 1 id pour "key"
- n fiches, et pour chacune:
  - 1 id "vieille fiche" deja connue par tous les processeurs
  - m nouvelles liaisons a ajouter, et pour chacune:
    - 1 id pour "variable"
    - 1 id pour "Term"
    - 1 id pout "shift"

6.2.2 Codage de son environnement.

Un objet sera transmis sous la forme suivante:

(S_id key		(F_id		(V_id		(T_id)Shift		)*		)*
	$\uparrow$		$\uparrow$				$\uparrow$		$\uparrow$	
	nbre fiches		nbre liaisons		$\overset{OU}{\longleftrightarrow}$		$\emptyset$		$\emptyset$	

Le processeur j calcule le collapse de l'objet reçu.

Chapitre 7

Conclusion.

Il n'y a plus de doute,

DyALog est parallélisable.

En effet, nous avons proposé 8 algorithmes qui, pour des grammaires et des machines données rendent tous un résulat très correct.

Pourtant, la parallélisation de DyALog nécéssite une bonne connaissance des conditions dans lesquelles elle s'appliquera. C'est à dire que si DyALog doit être parallélisé pour une grammaire du Langage Naturel (français par exemple), il faudra chercher un compromis entre l'algorithme et les machines susceptibles d'être utilisées.

Avant de commencer une parallélisation de DyALog pour une grammaire ou une machine particulière, il sera bon d'évaluer les performances futures à partir de mesures à faire sur l'exécution séquentielle, en utilisant les formules fournies[1].

De plus, comme l'étude présentée dans ce document se veut complète, si aucun algorithme parallèle ne s'avère utilisable pour une machine donnée, je crains qu'il ne faille changer de machine, ou se contenter de DyALog séquentiel.

Il faut tout de même tempérer l'euphorie que pourraient procurer les bons résultats obtenus par les algorithmes parallèles de DyALog. En effet le puriste ne sera pas satisfait: une machine à p processeurs n'atteint en aucun cas une accélération égale à p.

I.Debourges
LIFO, le 4 septembre 1998.

1. cf page 50

Annexe A

Résultats de simulation des algorithmes parallèles.

UNITÉS:

s	flops/seconde
l	flops
g	flops/mot
G1 .. S	secondes

LÉGENDE:

Tous les résultats fournis dans cette annexe sont issus des calculs réalisés avec les formules données pages 50-53.

TYPE DE GRAMMAIRE: **Langage naturel**

GRAMMAIRE TRAITEE: **Gate_CFG_169**

En rouge: les meilleurs résultats par constructeur
En vert: résultats pires qu'en séquentiel
Meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	0,76	5320000
R=	48,67	340690000
A=	0	0
S=	0,68	4760000
It=	2,38	16660000
Ia=	0,06	420000

tours de boucle en sequentiel:(n)	19462
applications en sequentiel:(n_appli)	34350
m:	10
nombre de liaisons:	3474
taille d'un objet:	2,536

temps sequentiel: 53,5 secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	5	5	5	5			5
	7,40E+07	2	1132	10,2	3	4	3	3	5	5	3
	7,40E+07	3	1496	9,5	2	3	3	2	3	3	2
	7,40E+07	4	1902	9,3	2	4	3	2	2	2	2
Origin 2000	1,01E+08	1	286	1,36	4	4	4	4			4
	1,01E+08	2	804	8,26	2	2	2	2	4	4	2
	1,01E+08	3	1313	8,36	2	2	2	1	2	2	1
	1,01E+08	4	1789	10,24	1	2	2	1	1	1	1
	1,01E+08	5	2474	11,06	1	3	3	1	1	1	1
	1,01E+08	6	2963	12,25	1	3	4	1	1	1	1
	1,01E+08	7	3867	14,28	2	4	6	1	1	1	1
Cray T3E	4,70E+07	1	86	2,14	8	8	8	8			8
	4,70E+07	2	269	2,61	4	4	4	4	8	8	4
	4,70E+07	3	296	2,11	3	3	3	3	4	4	3
	4,70E+07	4	357	1,77	2	3	3	2	3	3	2
	4,70E+07	8	506	1,64	2	2	3	1	1	1	1

	4,70E+07	9	552	1,57	2	2	3	1	1	1	1
	4,70E+07	16	751	1,66	1	3	6	1	1	1	1
	4,70E+07	20	880	1,63	1	3	8	1	1	1	1
	4,70E+07	24	1013	1,7	1	4	11	1	1	1	1
Cray T3D	1,20E+07	1	68	0,3	31	31	31	31			31
	1,20E+07	2	164	1	17	17	17	16	31	31	16
	1,20E+07	4	168	0,8	9	10	10	9	11	11	9
	1,20E+07	8	175	0,8	6	7	8	5	6	6	5
	1,20E+07	9	383	1,2	6	8	11	5	5	5	5
	1,20E+07	16	181	1	4	6	8	3	3	3	3
	1,20E+07	25	486	1,5	4	8	23	3	3	3	3
	1,20E+07	32	201	1,4	3	5	13	2	2	2	2
	1,20E+07	64	148	1,7	3	5	18	2	2	2	2
	1,20E+07	128	301	1,8	4	8	66	2	2	2	2
	1,20E+07	256	387	2,4	5	11	165	2	2	2	2
IBM SP2 (switch)	2,60E+07	1	244	1,3	14	15	14	14			14
	2,60E+07	2	1903	7,8	9	12	10	8	16	14	8
	2,60E+07	4	3583	8	7	15	15	5	6	5	5
	2,60E+07	8	5412	11,4	7	23	35	3	3	3	3
Pentium Pro NOW	6,10E+07	1	85	1	6	6	6	6			6
	6,10E+07	2	52745	484,5	20	55	38	12	24	8	12
	6,10E+07	4	139981	1128,5	49	186	184	14	18	5	14
	6,10E+07	8	539159	1994,1	184	880	1391	24	28	5	24
	6,10E+07	10	826054	2436,3	282	1431	2657	29	32	6	29
	6,10E+07	16	288444273	3614,6	92072	552229	1472496	5755	6139	617	5755
Digital 8400	2,40E+07	1	28	0,3	15	15	15	15			15
	2,40E+07	2	286	2,5	8	9	9	8	16	15	8
	2,40E+07	3	362	5,3	6	7	7	6	8	8	6
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	48	48	48	48			48
	7,70E+06	2	72966	128,8	211	581	396	118	234	67	118
	7,70E+06	4	369771	266,2	954	3763	3760	249	331	50	249

	7,70E+06	8	835281	322,3	2134	10588	16916	274	313	41	274
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	37	37	37	37			37
	1,00E+07	2	54	3,4	20	20	20	19	37	37	19
	1,00E+07	3	74	4,1	14	14	14	13	19	19	13
	1,00E+07	4	118	4,1	11	12	12	11	13	13	11
Hitachi SR2001	5,40E+06	1	31	0,2	68	68	68	68			68
	5,40E+06	2	1165	3	40	49	45	38	72	68	38
	5,40E+06	4	2299	3	29	53	53	21	28	25	21
	5,40E+06	8	3844	3,1	26	82	123	13	14	13	13
	5,40E+06	16	4638	3	25	109	276	8	9	8	8
	5,40E+06	32	6906	4,9	32	182	804	6	6	5	6
Convex Exemplar	1,05E+07	1	60	0,16	35	35	35	35			35
	1,05E+07	2	21373	8,3	58	138	98	38	75	39	38
	1,05E+07	4	64457	9,2	130	489	488	40	53	17	40
	1,05E+07	8	194476	11,3	367	1809	2890	51	58	12	51
Digital Alpha farm	1,01E+07	1	29	0,3	36	37	36	36			36
	1,01E+07	2	17202	81,1	53	120	86	36	70	40	36
	1,01E+07	3	34356	83	81	253	214	36	53	23	36
	1,01E+07	4	47109	81,3	103	376	376	33	44	17	34
Parsytec GC	1,93E+07	1	98	1	19	19	19	19			19
	1,93E+07	2	6309	113	17	30	23	13	26	20	14
	1,93E+07	4	23538	143	30	103	102	12	15	8	12
	1,93E+07	8	29080	254	37	158	244	7	8	5	8
	1,93E+07	16	224977	342	242	1382	3647	17	18	4	17
	1,93E+07	32	130527	658	185	987	4269	7	7	4	7
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	14	15	14	14			14
	2,60E+07	2	18759	183,6	22	51	36	15	29	16	15
	2,60E+07	4	39025	628,2	37	129	126	12	16	7	13
	2,60E+07	8	88795	1224,1	85	362	554	13	15	6	13

TYPE DE GRAMMAIRE: **Langage naturel**

GRAMMAIRE TRAITEE: **Gate_DCG_169**

En rouge: les meilleurs résultats par constructeur
En vert: résultats pires qu'en séquentiel
La meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	0,74	5180000
R=	56,54	395780000
A=	0	0
S=	0,61	4270000
It=	5,42	37940000
Ia=	0,09	630000

tours de boucle en sequentiel:(n)	21838
applications en sequentiel:(n_appli)	36317
m:	10
nombre de liaisons:	3415
taille d'un objet:	2,469

temps sequentiel: 65,72 secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	6	6	6	6			6
	7,40E+07	2	1132	10,2	4	4	4	3	6	6	3
	7,40E+07	3	1496	9,5	3	4	4	2	3	3	2
	7,40E+07	4	1902	9,3	3	4	4	2	3	2	2
Origin 2000	1,01E+08	1	286	1,36	4	5	4	4			4
	1,01E+08	2	804	8,26	3	3	3	2	5	4	2
	1,01E+08	3	1313	8,36	2	3	3	2	3	2	2
	1,01E+08	4	1789	10,24	2	3	3	1	2	2	1
	1,01E+08	5	2474	11,06	2	4	4	1	2	1	1
	1,01E+08	6	2963	12,25	2	4	5	1	1	1	1
	1,01E+08	7	3867	14,28	2	5	7	1	1	1	1
Cray T3E	4,70E+07	1	86	2,14	9	10	9	9			9
	4,70E+07	2	269	2,61	5	6	5	5	10	9	5
	4,70E+07	3	296	2,11	4	4	4	4	5	5	4
	4,70E+07	4	357	1,77	3	4	4	3	4	4	3
	4,70E+07	8	506	1,64	2	3	4	2	2	2	2

	4,70E+07	9	552	1,57	2	3	4	2	2	2	2
	4,70E+07	16	751	1,66	2	4	7	1	1	1	1
	4,70E+07	20	880	1,63	2	4	10	1	1	1	1
	4,70E+07	24	1013	1,7	2	4	13	1	1	1	1
Cray T3D	1,20E+07	1	68	0,3	37	37	37	37			37
	1,20E+07	2	164	1	21	21	21	20	37	37	20
	1,20E+07	4	168	0,8	12	13	13	12	15	14	12
	1,20E+07	8	175	0,8	8	9	10	7	8	8	7
	1,20E+07	9	383	1,2	8	11	14	7	7	7	7
	1,20E+07	16	181	1	6	8	11	5	5	5	5
	1,20E+07	25	486	1,5	6	11	27	5	5	5	5
	1,20E+07	32	201	1,4	5	7	17	4	4	4	4
	1,20E+07	64	148	1,7	5	7	22	4	4	4	4
	1,20E+07	128	301	1,8	5	10	75	3	3	3	3
	1,20E+07	256	387	2,4	7	14	187	3	3	3	3
IBM SP2 (switch)	2,60E+07	1	244	1,3	17	17	17	17			17
	2,60E+07	2	1903	7,8	11	14	13	10	19	17	10
	2,60E+07	4	3583	8	9	18	18	6	8	7	6
	2,60E+07	8	5412	11,4	8	27	40	4	4	4	4
Pentium Pro NOW	6,10E+07	1	85	1	7	7	7	7			7
	6,10E+07	2	52745	484,5	23	62	43	14	27	10	14
	6,10E+07	4	139981	1128,5	55	209	207	16	21	6	16
	6,10E+07	8	539159	1994,1	207	987	1561	27	31	6	28
	6,10E+07	10	826054	2436,3	316	1605	2981	33	37	7	33
	6,10E+07	16	288444273	3614,6	103312	619645	1652263	6458	6889	693	6458
Digital 8400	2,40E+07	1	28	0,3	19	19	19	19			18
	2,40E+07	2	286	2,5	10	11	11	10	19	18	10
	2,40E+07	3	362	5,3	8	9	8	7	10	10	7
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	58	58	58	58			58
	7,70E+06	2	72966	128,8	240	655	447	135	266	79	136
	7,70E+06	4	369771	266,2	1073	4225	4221	282	375	59	283

	7,70E+06	8	835281	322,3	2397	11883	18983	310	353	49	310
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	44	44	44	44			44
	1,00E+07	2	54	3,4	25	25	25	24	44	44	24
	1,00E+07	3	74	4,1	18	18	18	17	24	24	17
	1,00E+07	4	118	4,1	15	16	16	14	17	17	14
Hitachi SR2001	5,40E+06	1	31	0,2	82	82	82	82			82
	5,40E+06	2	1165	3	50	59	55	47	87	83	47
	5,40E+06	4	2299	3	36	64	64	28	35	32	28
	5,40E+06	8	3844	3,1	33	95	142	18	20	18	18
	5,40E+06	16	4638	3	32	126	313	13	13	12	13
	5,40E+06	32	6906	4,9	40	208	906	10	10	10	10
Convex Exemplar	1,05E+07	1	60	0,16	42	43	42	42			42
	1,05E+07	2	21373	8,3	68	157	112	45	87	47	45
	1,05E+07	4	64457	9,2	148	550	550	47	61	21	47
	1,05E+07	8	194476	11,3	414	2032	3245	59	67	15	59
Digital Alpha farm	1,01E+07	1	29	0,3	44	44	44	44			44
	1,01E+07	2	17202	81,1	62	137	99	43	82	48	43
	1,01E+07	3	34356	83	93	286	242	42	62	28	42
	1,01E+07	4	47109	81,3	117	424	424	40	52	21	40
Parsytec GC	1,93E+07	1	98	1	23	23	23	23			23
	1,93E+07	2	6309	113	20	35	27	16	31	24	17
	1,93E+07	4	23538	143	35	116	116	14	18	10	14
	1,93E+07	8	29080	254	43	178	274	9	10	6	10
	1,93E+07	16	224977	342	272	1551	4092	20	21	6	20
	1,93E+07	32	130527	658	208	1106	4789	9	9	5	9
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	17	17	17	17			17
	2,60E+07	2	18759	183,6	26	58	42	17	34	19	18
	2,60E+07	4	39025	628,2	42	145	142	15	19	9	15
	2,60E+07	8	88795	1224,1	96	406	622	15	17	7	16

TYPE DE GRAMMAIRE: **Langage naturel**

GRAMMAIRE TRAITEE: **Tag_3000**

En rouge: les meilleurs résultats par constructeur
En vert: résultats pires qu'en séquentiel

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	0,45	3150000
R=	12,83	89810000
A=	0	0
S=	3,39	23730000
It=	11,94	83580000
Ia=	0,04	280000

tours de boucle en sequentiel:(n)	16001
applications en sequentiel:(n_appli)	15001
m:	10
nombre de liaisons:	1
taille d'un objet:	2

temps sequentiel: 32,01 secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	2,8	2,8	2,8	2,8			2,8
	7,40E+07	2	1132	10,2	2,2	2,7	2,4	2,0	3,0	2,7	2,0
	7,40E+07	3	1496	9,5	2,0	2,9	2,7	1,8	2,1	1,9	1,8
	7,40E+07	4	1902	9,3	2,0	3,2	3,2	1,6	1,8	1,7	1,6
Origin 2000	1,01E+08	1	286	1,36	2,0	2,1	2,0	2,0			2,0
	1,01E+08	2	804	8,26	1,6	1,8	1,7	1,5	2,1	2,0	1,5
	1,01E+08	3	1313	8,36	1,5	2,0	1,9	1,3	1,5	1,4	1,3
	1,01E+08	4	1789	10,24	1,4	2,3	2,3	1,2	1,3	1,2	1,2
	1,01E+08	5	2474	11,06	1,5	2,8	3,1	1,1	1,2	1,1	1,1
	1,01E+08	6	2963	12,25	1,5	3,2	3,9	1,1	1,2	1,1	1,1
	1,01E+08	7	3867	14,28	1,7	4	5	1,1	1,1	1,0	1,1
Cray T3E	4,70E+07	1	86	2,14	4	4	4	4			4
	4,70E+07	2	269	2,61	3	3	3	3	4	4	3
	4,70E+07	3	296	2,11	3	3	3	3	3	3	3
	4,70E+07	4	357	1,77	3	3	3	2	3	3	2
	4,70E+07	8	506	1,64	2	3	4	2	2	2	2

	4,70E+07	9	552	1,57	2	3	4	2	2	2	2
	4,70E+07	16	751	1,66	2	4	6	2	2	2	2
	4,70E+07	20	880	1,63	2	4	8	2	2	2	2
	4,70E+07	24	1013	1,7	2	4	10	2	2	2	2
Cray T3D	1,20E+07	1	68	0,3	17	17	17	17			17
	1,20E+07	2	164	1	12	13	12	12	17	17	12
	1,20E+07	4	168	0,8	10	11	11	9	10	10	9
	1,20E+07	8	175	0,8	9	10	10	8	8	8	8
	1,20E+07	9	383	1,2	9	11	13	8	8	8	8
	1,20E+07	16	181	1	8	9	12	8	8	8	8
	1,20E+07	25	486	1,5	8	12	24	7	7	7	7
	1,20E+07	32	201	1,4	8	10	16	7	7	7	7
	1,20E+07	64	148	1,7	8	9	20	7	7	7	7
	1,20E+07	128	301	1,8	8	12	59	7	7	7	7
	1,20E+07	256	387	2,4	9	14	141	7	7	7	7
IBM SP2 (switch)	2,60E+07	1	244	1,3	8	8	8	8			8
	2,60E+07	2	1903	7,8	7	9	8	6	9	8	6
	2,60E+07	4	3583	8	7	13	13	5	5	5	5
	2,60E+07	8	5412	11,4	7	21	31	4	4	4	4
Pentium Pro NOW	6,10E+07	1	85	1	3	3	3	3			3
	6,10E+07	2	52745	484,5	16	44	30	9	18	5	10
	6,10E+07	4	139981	1128,5	40	152	151	11	15	4	12
	6,10E+07	8	539159	1994,1	150	720	1141	20	23	5	20
	6,10E+07	10	826054	2436,3	230	1172	2181	24	27	5	25
	6,10E+07	16	288444273	3614,6	75692	454013	1210628	4732	5048	508	4732
Digital 8400	2,40E+07	1	28	0,3	8	8	8	8			8
	2,40E+07	2	286	2,5	6	7	6	6	9	8	6
	2,40E+07	3	362	5,3	5	6	6	5	6	6	5
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	26	26	26	26			26
	7,70E+06	2	72966	128,8	171	475	323	95	179	42	95
	7,70E+06	4	369771	266,2	787	3095	3093	208	274	43	208

	7,70E+06	8	835281	322,3	1758	8706	13910	231	263	39	231
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	20	20	20	20			20
	1,00E+07	2	54	3,4	14	15	15	14	20	20	14
	1,00E+07	3	74	4,1	13	13	13	12	14	14	12
	1,00E+07	4	118	4,1	12	12	12	11	12	12	11
Hitachi SR2001	5,40E+06	1	31	0,2	37	37	37	37			37
	5,40E+06	2	1165	3	30	37	34	28	41	37	28
	5,40E+06	4	2299	3	28	49	49	23	25	23	23
	5,40E+06	8	3844	3,1	30	76	110	20	20	19	20
	5,40E+06	16	4638	3	31	100	238	18	18	17	18
	5,40E+06	32	6906	4,9	38	161	673	17	17	16	17
Convex Exemplar	1,05E+07	1	60	0,16	19	19	19	19			19
	1,05E+07	2	21373	8,3	46	111	79	30	52	22	30
	1,05E+07	4	64457	9,2	109	404	404	35	44	15	35
	1,05E+07	8	194476	11,3	306	1492	2381	46	52	14	46
Digital Alpha farm	1,01E+07	1	29	0,3	20	20	20	20			20
	1,01E+07	2	17202	81,1	42	97	69	28	48	23	28
	1,01E+07	3	34356	83	67	209	176	30	42	17	31
	1,01E+07	4	47109	81,3	87	311	311	30	37	15	30
Parsytec GC	1,93E+07	1	98	1	10	11	10	10			10
	1,93E+07	2	6309	113	13	24	18	10	16	11	10
	1,93E+07	4	23538	143	26	85	85	11	13	7	11
	1,93E+07	8	29080	254	32	130	201	8	9	6	9
	1,93E+07	16	224977	342	200	1135	2998	17	18	7	17
	1,93E+07	32	130527	658	147	802	3502	9	9	6	9
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	8	8	8	8			8
	2,60E+07	2	18759	183,6	17	41	29	11	20	9	12
	2,60E+07	4	39025	628,2	31	105	103	11	14	6	11
	2,60E+07	8	88795	1224,1	69	293	453	12	13	6	12

TYPE DE GRAMMAIRE: **Langage naturel**

GRAMMAIRE TRAITEE: **Dyck_151**

En rouge: les meilleurs résultats par constructeur
En vert: résultats pires qu'en séquentiel
Meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	3,04	21280000
R=	88,4	618800000
A=	0,79	5530000
S=	6,87	48090000
It=	1,4	9800000
Ia=	0,04	280000

tours de boucle en sequentiel:(n)	12162	
applications en sequentiel:(n_appli)	3,52E+05	
m:	10	
nombre de liaisons:	11552	
taille d'un objet:	4,85	
temps sequentiel:	105,8	secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	10	10	10	10			10
	7,40E+07	2	1132	10,2	5	6	5	5	10	10	5
	7,40E+07	3	1496	9,5	4	5	4	3	5	5	3
	7,40E+07	4	1902	9,3	3	4	4	3	3	3	3
Origin 2000	1,01E+08	1	286	1,36	7	7	7	7			7
	1,01E+08	2	804	8,26	4	4	4	4	7	7	4
	1,01E+08	3	1313	8,36	3	3	3	2	4	4	2
	1,01E+08	4	1789	10,24	2	3	3	2	2	2	2
	1,01E+08	5	2474	11,06	2	3	3	2	2	2	2
	1,01E+08	6	2963	12,25	2	3	4	1	2	1	1
	1,01E+08	7	3867	14,28	2	4	5	1	1	1	1
Cray T3E	4,70E+07	1	86	2,14	15	15	15	15			15
	4,70E+07	2	269	2,61	8	8	8	8	15	15	8
	4,70E+07	3	296	2,11	6	6	6	5	8	8	5
	4,70E+07	4	357	1,77	4	5	5	4	5	5	4
	4,70E+07	8	506	1,64	3	3	4	2	2	2	2

	4,70E+07	9	552	1,57	2	3	4	2	2	2	2
	4,70E+07	16	751	1,66	2	3	5	1	1	1	1
	4,70E+07	20	880	1,63	2	3	6	1	1	1	1
	4,70E+07	24	1013	1,7	2	3	8	1	1	1	1
Cray T3D	1,20E+07	1	68	0,3	59	59	59	59			59
	1,20E+07	2	164	1	31	31	31	30	59	59	30
	1,20E+07	4	168	0,8	17	17	17	15	20	20	15
	1,20E+07	8	175	0,8	10	11	11	8	9	9	8
	1,20E+07	9	383	1,2	9	11	13	7	8	8	7
	1,20E+07	16	181	1	6	7	9	4	5	5	4
	1,20E+07	25	486	1,5	6	9	18	3	3	3	3
	1,20E+07	32	201	1,4	5	6	11	3	3	3	3
	1,20E+07	64	148	1,7	4	6	14	2	2	2	2
	1,20E+07	128	301	1,8	4	7	43	1	1	1	1
	1,20E+07	256	387	2,4	6	10	107	1	1	1	1
IBM SP2 (switch)	2,60E+07	1	244	1,3	27	27	27	27			27
	2,60E+07	2	1903	7,8	15	17	16	14	28	27	14
	2,60E+07	4	3583	8	9	15	15	7	10	9	7
	2,60E+07	8	5412	11,4	7	18	26	4	5	4	4
Pentium Pro NOW	6,10E+07	1	85	1	12	12	12	12			12
	6,10E+07	2	52745	484,5	17	45	34	11	23	13	12
	6,10E+07	4	139981	1128,5	34	144	142	11	15	6	11
	6,10E+07	8	539159	1994,1	123	607	924	17	19	5	17
	6,10E+07	10	826054	2436,3	188	969	1731	20	22	6	20
	6,10E+07	16	288444273	3614,6	57563	345217	920295	3598	3839	388	3599
Digital 8400	2,40E+07	1	28	0,3	29	29	29	29			29
	2,40E+07	2	286	2,5	15	16	16	15	29	29	15
	2,40E+07	3	362	5,3	11	12	11	10	15	15	10
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	92	92	92	92			92
	7,70E+06	2	72966	128,8	164	410	293	104	209	104	105
	7,70E+06	4	369771	266,2	616	2417	2412	171	229	53	172

	7,70E+06	8	835281	322,3	1352	6699	10649	180	205	35	180
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	70	70	70	70			70
	1,00E+07	2	54	3,4	37	37	37	36	70	70	36
	1,00E+07	3	74	4,1	26	26	26	24	36	36	24
	1,00E+07	4	118	4,1	20	21	21	18	24	24	18
Hitachi SR2001	5,40E+06	1	31	0,2	130	130	130	130			130
	5,40E+06	2	1165	3	71	76	74	67	133	131	67
	5,40E+06	4	2299	3	42	59	58	35	46	45	35
	5,40E+06	8	3844	3,1	30	66	92	19	21	20	19
	5,40E+06	16	4638	3	25	78	182	11	11	10	11
	5,40E+06	32	6906	4,9	27	122	511	6	7	6	6
Convex Exemplar	1,05E+07	1	60	0,16	67	67	67	67			67
	1,05E+07	2	21373	8,3	60	110	85	46	92	70	46
	1,05E+07	4	64457	9,2	94	319	319	36	48	25	36
	1,05E+07	8	194476	11,3	237	1140	1815	37	43	14	37
Digital Alpha farm	1,01E+07	1	29	0,3	70	70	70	70			70
	1,01E+07	2	17202	81,1	58	106	85	46	91	72	46
	1,01E+07	3	34356	83	68	185	160	38	57	38	38
	1,01E+07	4	47109	81,3	78	259	258	33	43	26	33
Parsytec GC	1,93E+07	1	98	1	37	37	37	37			37
	1,93E+07	2	6309	113	23	37	32	21	41	37	21
	1,93E+07	4	23538	143	26	81	80	14	18	13	14
	1,93E+07	8	29080	254	30	125	178	8	9	7	8
	1,93E+07	16	224977	342	161	903	2316	13	13	5	13
	1,93E+07	32	130527	658	147	707	2754	6	6	4	6
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	27	27	27	27			27
	2,60E+07	2	18759	183,6	23	47	38	18	37	28	19
	2,60E+07	4	39025	628,2	30	119	116	13	17	11	13
	2,60E+07	8	88795	1224,1	65	310	426	11	13	8	12

TYPE DE GRAMMAIRE: **Langage naturel**

GRAMMAIRE TRAITEE: **Bio_123**

En rouge: les meilleurs résultats par constructeur
En vert: résultats pires qu'en séquentiel
La meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

Les résultats des calculs sont en secondes.

PARAMETRES:

	secondes	flops
C=	3,3	23100000
R=	196,99	1378930000
A=	0	0
S=	6,46	45220000
It=	5,74	40180000
Ia=	0,23	1610000

tours de boucle en sequentiel:(n)	57370
applications en sequentiel:(n_appli)	3,68E+05
m:	10
nombre de liaisons:	55749
taille d'un objet:	4,92

temps sequentiel: 219,5 secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	20	20	20	20			20
	7,40E+07	2	1132	10,2	11	13	12	11	21	20	11
	7,40E+07	3	1496	9,5	9	12	11	7	11	10	7
	7,40E+07	4	1902	9,3	7	12	12	6	8	7	6
Origin 2000	1,01E+08	1	286	1,36	15	15	15	15			15
	1,01E+08	2	804	8,26	8	9	9	8	15	15	8
	1,01E+08	3	1313	8,36	6	8	8	5	8	8	5
	1,01E+08	4	1789	10,24	5	9	8	4	6	5	4
	1,01E+08	5	2474	11,06	5	10	11	4	4	4	4
	1,01E+08	6	2963	12,25	5	11	13	3	4	3	3
	1,01E+08	7	3867	14,28	5	14	19	3	3	3	3
Cray T3E	4,70E+07	1	86	2,14	32	32	32	32			32
	4,70E+07	2	269	2,61	17	18	17	16	32	32	16
	4,70E+07	3	296	2,11	12	13	13	11	16	16	11
	4,70E+07	4	357	1,77	9	11	11	9	11	11	9
	4,70E+07	8	506	1,64	6	8	10	5	5	5	5

	4,70E+07	9	552	1,57	5	8	11	4	5	5	4
	4,70E+07	16	751	1,66	4	9	18	3	3	3	3
	4,70E+07	20	880	1,63	4	10	25	2	3	2	2
	4,70E+07	24	1013	1,7	4	11	33	2	2	2	2
Cray T3D	1,20E+07	1	68	0,3	124	125	124	124			124
	1,20E+07	2	164	1	66	67	66	64	125	124	64
	1,20E+07	4	168	0,8	36	38	38	34	44	44	34
	1,20E+07	8	175	0,8	21	25	27	19	21	21	19
	1,20E+07	9	383	1,2	21	29	35	17	19	18	17
	1,20E+07	16	181	1	14	19	27	11	11	11	11
	1,20E+07	25	486	1,5	13	27	69	8	9	8	8
	1,20E+07	32	201	1,4	11	17	41	7	7	7	7
	1,20E+07	64	148	1,7	10	16	55	5	5	5	5
	1,20E+07	128	301	1,8	13	25	196	4	4	4	4
	1,20E+07	256	387	2,4	22	40	494	4	4	4	4
IBM SP2 (switch)	2,60E+07	1	244	1,3	58	58	58	58			58
	2,60E+07	2	1903	7,8	34	43	39	32	62	58	32
	2,60E+07	4	3583	8	24	49	48	18	23	20	18
	2,60E+07	8	5412	11,4	22	71	106	10	11	10	10
Pentium Pro NOW	6,10E+07	1	85	1	24	25	24	24			24
	6,10E+07	2	52745	484,5	65	173	121	38	78	32	41
	6,10E+07	4	139981	1128,5	154	585	574	43	59	18	46
	6,10E+07	8	539159	1994,1	576	2683	4177	75	87	21	77
	6,10E+07	10	826054	2436,3	882	4342	7939	91	102	23	93
	6,10E+07	16	288444273	3614,6	271533	1628097	4340825	16973	18105	1827	16975
Digital 8400	2,40E+07	1	28	0,3	62	62	62	62			62
	2,40E+07	2	286	2,5	33	35	34	32	63	62	32
	2,40E+07	3	362	5,3	23	26	25	22	32	32	22
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	194	195	194	194			194
	7,70E+06	2	72966	128,8	649	1756	1208	374	746	252	378
	7,70E+06	4	369771	266,2	2839	11170	11151	748	999	169	753

	7,70E+06	8	835281	322,3	6338	31333	49968	817	935	133	820
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	149	149	149	149			149
	1,00E+07	2	54	3,4	78	79	79	77	149	149	77
	1,00E+07	3	74	4,1	55	56	56	53	77	77	53
	1,00E+07	4	118	4,1	43	46	46	41	53	52	41
Hitachi SR2001	5,40E+06	1	31	0,2	276	276	276	276			276
	5,40E+06	2	1165	3	156	182	169	148	288	277	148
	5,40E+06	4	2299	3	103	177	177	81	105	98	81
	5,40E+06	8	3844	3,1	87	252	374	46	52	46	46
	5,40E+06	16	4638	3	80	328	820	28	29	26	27
	5,40E+06	32	6906	4,9	102	545	2378	19	19	17	18
Convex Exemplar	1,05E+07	1	60	0,16	142	142	142	142			142
	1,05E+07	2	21373	8,3	191	425	308	131	259	154	131
	1,05E+07	4	64457	9,2	393	1451	1451	127	167	62	127
	1,05E+07	8	194476	11,3	1088	5341	8528	154	176	39	154
Digital Alpha farm	1,01E+07	1	29	0,3	148	148	148	148			147
	1,01E+07	2	17202	81,1	177	382	282	126	250	159	128
	1,01E+07	3	34356	83	253	771	653	118	177	88	120
	1,01E+07	4	47109	81,3	316	1134	1130	109	144	63	110
Parsytec GC	1,93E+07	1	98	1	77	78	77	77			77
	1,93E+07	2	6309	113	61	105	85	50	99	81	51
	1,93E+07	4	23538	143	98	322	318	40	53	32	41
	1,93E+07	8	29080	254	125	503	751	26	29	18	26
	1,93E+07	16	224977	342	752	4145	10813	53	57	17	54
	1,93E+07	32	130527	658	692	3127	12779	26	27	15	26
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	58	58	58	58			58
	2,60E+07	2	18759	183,6	73	164	121	51	103	63	53
	2,60E+07	4	39025	628,2	123	427	414	42	58	30	46
	2,60E+07	8	88795	1224,1	298	1196	1744	45	53	26	48

TYPE DE GRAMMAIRE: **Faiblement ambiguë**

En rouge: les meilleurs résultats

GRAMMAIRE TRAITEE: **Hanoï_18**

En vert: résultats pires qu'en séquentiel

La meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	0,08	560000
R=	77,21	540470000
A=	0,18	1260000
S=	0,01	70000
It=	0,02	140000
Ia=	0	0

tours de boucle en sequentiel:(n)	41
applications en sequentiel:(n_appli)	1,38E+02
m:	10
nombre de liaisons:	11762
taille d'un objet:	862,63

temps sequentiel: 83,07 secondes

ALGORITHMES / MACHINES	s	p	l	g	G1	G21	G22	I	I'	I²	S
SGI Power Challenge	7,40E+07	1	226	0,5	7	7	7	7			7
	7,40E+07	2	1132	10,2	4	4	4	4	7	7	4
	7,40E+07	3	1496	9,5	2	2	2	2	4	4	2
	7,40E+07	4	1902	9,3	2	2	2	2	2	2	2
Origin 2000	1,01E+08	1	286	1,36	5	5	5	5			5
	1,01E+08	2	804	8,26	3	3	3	3	5	5	3
	1,01E+08	3	1313	8,36	2	2	2	2	3	3	2
	1,01E+08	4	1789	10,24	1	1	1	1	2	2	1
	1,01E+08	5	2474	11,06	1	1	1	1	1	1	1
	1,01E+08	6	2963	12,25	1	1	1	1	1	1	1
	1,01E+08	7	3867	14,28	1	1	1	1	1	1	1
Cray T3E	4,70E+07	1	86	2,14	12	12	12	12			12
	4,70E+07	2	269	2,61	6	6	6	6	12	12	6
	4,70E+07	3	296	2,11	4	4	4	4	6	6	4
	4,70E+07	4	357	1,77	3	3	3	3	4	4	3
	4,70E+07	8	506	1,64	1	1	1	1	2	2	1

	4,70E+07	9	552	1,57	1	1	1	1	1	1	1
	4,70E+07	16	751	1,66	1	1	1	1	1	1	1
	4,70E+07	20	880	1,63	1	1	1	1	1	1	1
	4,70E+07	24	1013	1,7	1	1	1	0,5	1	1	0,5
Cray T3D	1,20E+07	1	68	0,3	45	45	45	45			45
	1,20E+07	2	164	1	23	23	23	23	45	45	23
	1,20E+07	4	168	0,8	11	11	11	11	15	15	11
	1,20E+07	8	175	0,8	6	6	6	6	6	6	6
	1,20E+07	9	383	1,2	5	5	5	5	6	6	5
	1,20E+07	16	181	1	3	3	3	3	3	3	3
	1,20E+07	25	486	1,5	2	2	2	2	2	2	2
	1,20E+07	32	201	1,4	2	2	2	1	1	1	1
	1,20E+07	64	148	1,7	1	1	1	1	1	1	1
	1,20E+07	128	301	1,8	1	1	1	0,4	0,4	0,4	0,4
	1,20E+07	256	387	2,4	2	2	2	0,2	0,2	0,2	0,2
IBM SP2 (switch)	2,60E+07	1	244	1,3	21	21	21	21			21
	2,60E+07	2	1903	7,8	10	10	10	10	21	21	10
	2,60E+07	4	3583	8	5	5	5	5	7	7	5
	2,60E+07	8	5412	11,4	3	3	3	3	3	3	3
Pentium Pro NOW	6,10E+07	1	85	1	9	9	9	9			9
	6,10E+07	2	52745	484,5	5	6	5	5	9	9	5
	6,10E+07	4	139981	1128,5	4	8	6	3	4	4	3
	6,10E+07	8	539159	1994,1	10	18	16	2	3	2	2
	6,10E+07	10	826054	2436,3	14	26	23	2	3	2	3
	6,10E+07	16	288444273	3614,6	226	1210	3141	15	16	4	15
Digital 8400	2,40E+07	1	28	0,3	23	23	23	23			23
	2,40E+07	2	286	2,5	11	11	11	11	23	23	11
	2,40E+07	3	362	5,3	8	8	8	8	11	11	8
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	70	70	70	70			70
	7,70E+06	2	72966	128,8	36	39	38	36	72	71	36
	7,70E+06	4	369771	266,2	23	35	32	19	26	25	20

	7,70E+06	8	835281	322,3	24	50	59	11	12	12	11
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	54	54	54	54			54
	1,00E+07	2	54	3,4	27	27	27	27	54	54	27
	1,00E+07	3	74	4,1	18	18	18	18	27	27	18
	1,00E+07	4	118	4,1	14	14	14	14	18	18	14
Hitachi SR2001	5,40E+06	1	31	0,2	100	100	100	100			100
	5,40E+06	2	1165	3	50	50	50	50	101	100	50
	5,40E+06	4	2299	3	25	25	25	25	34	34	25
	5,40E+06	8	3844	3,1	13	13	13	13	14	14	13
	5,40E+06	16	4638	3	7	7	7	6	7	7	6
	5,40E+06	32	6906	4,9	4	5	6	3	3	3	3
Convex Exemplar	1,05E+07	1	60	0,16	52	52	52	52			52
	1,05E+07	2	21373	8,3	26	26	26	26	52	52	26
	1,05E+07	4	64457	9,2	13	14	14	13	17	17	13
	1,05E+07	8	194476	11,3	8	11	13	7	8	7	7
Digital Alpha farm	1,01E+07	1	29	0,3	54	54	54	54			54
	1,01E+07	2	17202	81,1	27	28	28	27	54	54	27
	1,01E+07	3	34356	83	19	20	20	18	27	27	18
	1,01E+07	4	47109	81,3	15	16	16	14	18	18	14
Parsytec GC	1,93E+07	1	98	1	28	28	28	28			28
	1,93E+07	2	6309	113	14	15	15	14	29	28	14
	1,93E+07	4	23538	143	8	9	9	7	10	10	7
	1,93E+07	8	29080	254	7	10	9	4	5	4	4
	1,93E+07	16	224977	342	12	19	21	2	3	3	2
	1,93E+07	32	130527	658	39	50	51	2	2	2	2
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	21	21	21	21			21
	2,60E+07	2	18759	183,6	11	11	11	11	21	21	11
	2,60E+07	4	39025	628,2	8	12	10	6	8	8	6
	2,60E+07	8	88795	1224,1	14	25	20	4	5	5	5

TYPE DE GRAMMAIRE: **Faiblement ambiguë**

GRAMMAIRE TRAITEE: **Append_8000 avec requete indéterministe**

En rouge: les meilleurs résultats
En vert: résultats pires qu'en séquentiel
Meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	0	0
R=	0	0
A=	4,45	31150000
S=	0	0
It=	0,01	70000
Ia=	0	0

tours de boucle en sequentiel:(n)	14
applications en sequentiel:(n_appli)	1,8E+01
m:	10
nombre de liaisons:	6
taille d’un objet:	3,29
temps sequentiel:	53,5 secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I’	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	0,4	0,4	0,4	0,4			0,4
	7,40E+07	2	1132	10,2	0,2	0,2	0,2	0,2	0,4	0,4	0,2
	7,40E+07	3	1496	9,5	0,14	0,14	0,14	0,14	0,2	0,2	0,14
	7,40E+07	4	1902	9,3	0,11	0,11	0,11	0,11	0,14	0,14	0,11
Origin 2000	1,01E+08	1	286	1,36	0,3	0,3	0,3	0,3			0,3
	1,01E+08	2	804	8,26	0,2	0,2	0,2	0,2	0,3	0,3	0,2
	1,01E+08	3	1313	8,36	0,10	0,10	0,10	0,10	0,2	0,2	0,10
	1,01E+08	4	1789	10,24	0,08	0,08	0,08	0,08	0,10	0,10	0,08
	1,01E+08	5	2474	11,06	0,06	0,06	0,06	0,06	0,08	0,08	0,06
	1,01E+08	6	2963	12,25	0,05	0,05	0,05	0,05	0,06	0,06	0,05
	1,01E+08	7	3867	14,28	0,05	0,05	0,05	0,04	0,05	0,05	0,04
Cray T3E	4,70E+07	1	86	2,14	0,7	0,7	0,7	0,7			0,7
	4,70E+07	2	269	2,61	0,3	0,3	0,3	0,3	0,7	0,7	0,3
	4,70E+07	3	296	2,11	0,2	0,2	0,2	0,2	0,3	0,3	0,2
	4,70E+07	4	357	1,77	0,2	0,2	0,2	0,2	0,2	0,2	0,2
	4,70E+07	8	506	1,64	0,08	0,09	0,09	0,08	0,10	0,10	0,08

	4,70E+07	9	552	1,57	0,08	0,08	0,08	0,08	0,08	0,08	0,08
	4,70E+07	16	751	1,66	0,04	0,04	0,05	0,04	0,05	0,05	0,04
	4,70E+07	20	880	1,63	0,03	0,04	0,04	0,03	0,04	0,04	0,03
	4,70E+07	24	1013	1,7	0,03	0,03	0,04	0,03	0,03	0,03	0,03
Cray T3D	1,20E+07	1	68	0,3	2,6	2,6	2,6	2,6			2,6
	1,20E+07	2	164	1	1,3	1,3	1,3	1,3	2,6	2,6	1,3
	1,20E+07	4	168	0,8	0,7	0,7	0,7	0,7	0,9	0,9	0,7
	1,20E+07	8	175	0,8	0,3	0,3	0,3	0,3	0,4	0,4	0,3
	1,20E+07	9	383	1,2	0,3	0,3	0,3	0,3	0,3	0,3	0,3
	1,20E+07	16	181	1	0,2	0,2	0,2	0,2	0,2	0,2	0,2
	1,20E+07	25	486	1,5	0,11	0,11	0,12	0,11	0,11	0,11	0,11
	1,20E+07	32	201	1,4	0,09	0,09	0,09	0,09	0,09	0,09	0,09
	1,20E+07	64	148	1,7	0,05	0,05	0,06	0,05	0,05	0,05	0,05
	1,20E+07	128	301	1,8	0,03	0,03	0,07	0,03	0,03	0,03	0,03
	1,20E+07	256	387	2,4	0,02	0,02	0,13	0,02	0,02	0,02	0,02
IBM SP2 (switch)	2,60E+07	1	244	1,3	1,2	1,2	1,2	1,2			1,2
	2,60E+07	2	1903	7,8	0,6	0,6	0,6	0,6	1,2	1,2	0,6
	2,60E+07	4	3583	8	0,3	0,3	0,3	0,3	0,4	0,4	0,3
	2,60E+07	8	5412	11,4	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Pentium Pro NOW	6,10E+07	1	85	1	0,5	0,5	0,5	0,5			0,5
	6,10E+07	2	52745	484,5	0,3	0,3	0,3	0,3	0,5	0,5	0,3
	6,10E+07	4	139981	1128,5	0,2	0,3	0,3	0,1	0,2	0,2	0,1
	6,10E+07	8	539159	1994,1	0,2	0,7	1,1	0,1	0,1	0,1	0,1
	6,10E+07	10	826054	2436,3	0,3	1,1	2	0,1	0,1	0,1	0,1
	6,10E+07	16	288444273	3614,6	66	397	1059	4	4	0,5	4
Digital 8400	2,40E+07	1	28	0,3	1,3	1,3	1,3	1,3			1,3
	2,40E+07	2	286	2,5	0,7	0,7	0,7	0,7	1,3	1,3	0,7
	2,40E+07	3	362	5,3	0,4	0,4	0,4	0,4	0,7	0,7	0,4
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	4,1	4,1	4,1	4,1			4,1
	7,70E+06	2	72966	128,8	2,2	2,4	2,3	2,1	4,2	4,1	2,1
	7,70E+06	4	369771	266,2	1,7	3,7	3,7	1,2	1,6	1,4	1,2

	7,70E+06	8	835281	322,3	2	8,1	12,7	0,7	0,8	0,6	0,7
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	3,1	3,1	3,1	3,1			3,1
	1,00E+07	2	54	3,4	1,6	1,6	1,6	1,6	3,1	3,1	1,6
	1,00E+07	3	74	4,1	1,0	1,0	1,0	1,0	1,6	1,6	1,0
	1,00E+07	4	118	4,1	0,8	0,8	0,8	0,8	1,0	1,0	0,8
Hitachi SR2001	5,40E+06	1	31	0,2	5,8	5,8	5,8	5,8			5,8
	5,40E+06	2	1165	3	2,9	2,9	2,9	2,9	5,8	5,8	2,9
	5,40E+06	4	2299	3	1,5	1,5	1,5	1,5	1,9	1,9	1,5
	5,40E+06	8	3844	3,1	0,7	0,8	0,8	0,7	0,8	0,8	0,7
	5,40E+06	16	4638	3	0,4	0,4	0,6	0,4	0,4	0,4	0,4
	5,40E+06	32	6906	4,9	0,2	0,3	0,8	0,2	0,2	0,2	0,2
Convex Exemplar	1,05E+07	1	60	0,16	3,0	3,0	3,0	3,0			3,0
	1,05E+07	2	21373	8,3	1,5	1,6	1,5	1,5	3,0	3,0	1,5
	1,05E+07	4	64457	9,2	0,8	1,1	1,1	0,8	1,0	1,0	0,8
	1,05E+07	8	194476	11,3	0,6	1,7	2,5	0,4	0,5	0,4	0,4
Digital Alpha farm	1,01E+07	1	29	0,3	3,1	3,1	3,1	3,1			3,1
	1,01E+07	2	17202	81,1	1,6	1,6	1,6	1,6	3,1	3,1	1,6
	1,01E+07	3	34356	83	1,1	1,2	1,2	1,1	1,6	1,6	1,1
	1,01E+07	4	47109	81,3	0,8	1,0	1,0	0,8	1,1	1,0	0,8
Parsytec GC	1,93E+07	1	98	1	1,6	1,6	1,6	1,6			1,6
	1,93E+07	2	6309	113	0,8	0,8	0,8	0,8	1,6	1,6	0,8
	1,93E+07	4	23538	143	0,4	0,5	0,5	0,4	0,5	0,5	0,4
	1,93E+07	8	29080	254	0,2	0,3	0,4	0,2	0,2	0,2	0,2
	1,93E+07	16	224977	342	0,3	1,1	2,7	0,1	0,1	0,1	0,1
	1,93E+07	32	130527	658	0,2	0,8	3,1	0,1	0,1	0,1	0,1
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	1,2	1,2	1,2	1,2			1,2
	2,60E+07	2	18759	183,6	0,6	0,6	0,6	0,6	1,2	1,2	0,6
	2,60E+07	4	39025	628,2	0,3	0,4	0,4	0,3	0,4	0,4	0,3
	2,60E+07	8	88795	1224,1	0,2	0,4	0,6	0,2	0,2	0,2	0,2

TYPE DE GRAMMAIRE: **Déterministe**

GRAMMAIRE TRAITEE: **Palin_400**

En rouge: les meilleurs résultats
En vert: résultats pires qu'en séquentiel
Meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	2,84	19880000
R=	23,95	167650000
A=	19,48	136360000
S=	21,72	152040000
It=	11,54	80780000
Ia=	0,3	2100000

tours de boucle en sequentiel:(n)	81405	
applications en sequentiel:(n_appli)	1,6E+05	
m:	10	
nombre de liaisons:	79801	
taille d'un objet:	4,94	
temps sequentiel:	112,6	secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	8	8	8	8			8
	7,40E+07	2	1132	10,2	6	8	7	5	9	8	5
	7,40E+07	3	1496	9,5	5	10	9	4	5	4	4
	7,40E+07	4	1902	9,3	5	12	12	3	4	3	3
Origin 2000	1,01E+08	1	286	1,36	6	6	6	6			6
	1,01E+08	2	804	8,26	4	5	5	4	6	6	4
	1,01E+08	3	1313	8,36	4	7	6	3	4	3	3
	1,01E+08	4	1789	10,24	4	8	8	2	3	2	2
	1,01E+08	5	2474	11,06	4	11	12	2	3	2	2
	1,01E+08	6	2963	12,25	4	13	16	2	2	2	2
	1,01E+08	7	3867	14,28	5	17	24	2	2	2	2
Cray T3E	4,70E+07	1	86	2,14	12	12	12	12			12
	4,70E+07	2	269	2,61	8	9	8	7	12	12	7
	4,70E+07	3	296	2,11	6	7	7	5	7	7	5
	4,70E+07	4	357	1,77	5	7	7	4	5	5	4
	4,70E+07	8	506	1,64	4	8	11	3	3	3	3

	4,70E+07	9	552	1,57	4	8	12	3	3	3	3
	4,70E+07	16	751	1,66	4	11	24	2	2	2	2
	4,70E+07	20	880	1,63	4	13	33	2	2	2	2
	4,70E+07	24	1013	1,7	5	15	45	2	2	2	2
Cray T3D	1,20E+07	1	68	0,3	47	47	47	47			47
	1,20E+07	2	164	1	29	31	30	27	48	47	27
	1,20E+07	4	168	0,8	19	23	23	17	20	20	17
	1,20E+07	8	175	0,8	15	20	23	12	13	12	12
	1,20E+07	9	383	1,2	16	27	37	12	12	12	11
	1,20E+07	16	181	1	13	19	31	9	9	9	9
	1,20E+07	25	486	1,5	15	34	94	9	9	8	9
	1,20E+07	32	201	1,4	13	21	55	8	8	8	8
	1,20E+07	64	148	1,7	14	21	77	8	7	7	7
	1,20E+07	128	301	1,8	19	35	278	7	7	7	7
	1,20E+07	256	387	2,4	32	56	701	7	7	7	7
IBM SP2 (switch)	2,60E+07	1	244	1,3	22	23	22	22			22
	2,60E+07	2	1903	7,8	19	31	25	15	28	22	15
	2,60E+07	4	3583	8	20	54	54	11	13	10	11
	2,60E+07	8	5412	11,4	24	93	143	8	8	6	8
Pentium Pro NOW	6,10E+07	1	85	1	9	9	9	9			9
	6,10E+07	2	52745	484,5	79	226	153	42	86	19	45
	6,10E+07	4	139981	1128,5	213	799	784	56	76	18	59
	6,10E+07	8	539159	1994,1	814	3755	5874	104	120	26	107
	6,10E+07	10	826054	2436,3	1249	6096	11200	127	143	31	130
	6,10E+07	16	288444273	3614,6	385291	2310087	6159304	24082	25689	2592	24085
Digital 8400	2,40E+07	1	28	0,3	23	23	23	23			23
	2,40E+07	2	286	2,5	15	17	16	14	24	23	14
	2,40E+07	3	362	5,3	12	15	15	11	14	13	11
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	73	74	73	73			73
	7,70E+06	2	72966	128,8	821	2377	1599	431	857	156	437
	7,70E+06	4	369771	266,2	3979	15755	15728	1014	1353	175	1021

	7,70E+06	8	835281	322,3	8969	44372	70813	1137	1300	162	1141
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	56	56	56	56			56
	1,00E+07	2	54	3,4	34	35	34	32	56	56	32
	1,00E+07	3	74	4,1	26	28	28	24	32	32	24
	1,00E+07	4	118	4,1	23	27	26	21	24	24	20
Hitachi SR2001	5,40E+06	1	31	0,2	104	104	104	104			104
	5,40E+06	2	1165	3	79	115	97	68	121	105	68
	5,40E+06	4	2299	3	75	180	180	46	56	46	46
	5,40E+06	8	3844	3,1	89	322	495	34	36	29	33
	5,40E+06	16	4638	3	98	449	1147	25	26	22	25
	5,40E+06	32	6906	4,9	137	764	3365	22	22	19	21
Convex Exemplar	1,05E+07	1	60	0,16	54	54	54	54			53
	1,05E+07	2	21373	8,3	198	530	364	114	219	70	114
	1,05E+07	4	64457	9,2	521	2022	2021	144	190	40	144
	1,05E+07	8	194476	11,3	1526	7559	12081	202	230	36	202
Digital Alpha farm	1,01E+07	1	29	0,3	56	56	56	56			55
	1,01E+07	2	17202	81,1	175	458	316	103	200	72	106
	1,01E+07	3	34356	83	309	1034	867	118	175	49	120
	1,01E+07	4	47109	81,3	411	1561	1555	117	155	40	119
Parsytec GC	1,93E+07	1	98	1	29	30	29	29			29
	1,93E+07	2	6309	113	46	104	75	31	60	34	33
	1,93E+07	4	23538	143	119	428	422	38	49	19	39
	1,93E+07	8	29080	254	168	684	1036	27	31	15	29
	1,93E+07	16	224977	342	1063	5849	15310	72	77	19	73
	1,93E+07	32	130527	658	982	4380	18075	36	37	20	36
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	22	23	22	22			22
	2,60E+07	2	18759	183,6	74	197	136	43	86	30	46
	2,60E+07	4	39025	628,2	160	560	541	46	63	23	50
	2,60E+07	8	88795	1224,1	417	1619	2396	57	67	29	61

TYPE DE GRAMMAIRE: **Déterministe**

GRAMMAIRE TRAITEE: **Append_6000**

En rouge: les meilleurs résultats
En vert: résultats pires qu'en séquentiel
Meilleure accélération pour cette grammaire

Machine utilisée SPARC Station 4 7E+06 flops/seconde

PARAMETRES:

Les résultats des calculs sont en secondes.

	secondes	flops
C=	0,51	3570000
R=	0	0
A=	19,51	136570000
S=	0	0
It=	1,66	11620000
Ia=	0,02	140000

tours de boucle en sequentiel:(n)	6004
applications en sequentiel:(n_appli)	6,0E+03
m:	10
nombre de liaisons:	81288
taille d'un objet:	42,62
temps sequentiel:	22,79 secondes

ALGORITHMES	s	p	l	g	G1	G21	G22	I	I'	I²	S
MACHINES											
SGI Power Challenge	7,40E+07	1	226	0,5	2	2	2	2			2
	7,40E+07	2	1132	10,2	1,3	1,5	1,4	1,2	2	2	1,2
	7,40E+07	3	1496	9,5	1,0	1,4	1,3	0,9	1,2	1,1	0,9
	7,40E+07	4	1902	9,3	0,9	1,5	1,4	0,7	0,9	0,8	0,7
Origin 2000	1,01E+08	1	286	1,36	2	2	2	2			2
	1,01E+08	2	804	8,26	0,9	1,0	1,0	0,8	2	2	0,9
	1,01E+08	3	1313	8,36	0,7	1,0	0,9	0,6	0,9	0,8	0,6
	1,01E+08	4	1789	10,24	0,7	1,1	1,0	0,5	0,6	0,6	0,5
	1,01E+08	5	2474	11,06	0,7	1,3	1,3	0,4	0,5	0,5	0,5
	1,01E+08	6	2963	12,25	0,7	1,5	2	0,4	0,5	0,4	0,4
	1,01E+08	7	3867	14,28	0,8	2	2	0,4	0,4	0,4	0,4
Cray T3E	4,70E+07	1	86	2,14	3	3	3	3			3
	4,70E+07	2	269	2,61	2	2	2	2	3	3	2
	4,70E+07	3	296	2,11	1,4	1,5	1,4	1,3	2	2	1,3
	4,70E+07	4	357	1,77	1,1	1,3	1,3	1,0	1,3	1,3	1,0
	4,70E+07	8	506	1,64	0,8	1,1	1,3	0,6	0,7	0,7	0,6

	4,70E+07	9	552	1,57	0,8	1,1	1,4	0,6	0,6	0,6	0,6
	4,70E+07	16	751	1,66	0,7	1,3	2	0,5	0,5	0,5	0,4
	4,70E+07	20	880	1,63	0,8	1,4	3	0,4	0,4	0,4	0,4
	4,70E+07	24	1013	1,7	0,8	2	4	0,4	0,4	0,4	0,4
Cray T3D	1,20E+07	1	68	0,3	13	13	13	13			13
	1,20E+07	2	164	1	7	7	7	7	13	13	7
	1,20E+07	4	168	0,8	4	5	5	4	5	5	4
	1,20E+07	8	175	0,8	3	3	4	2	3	3	2
	1,20E+07	9	383	1,2	3	4	4	2	2	2	2
	1,20E+07	16	181	1	2	3	4	2	2	2	2
	1,20E+07	25	486	1,5	3	4	9	1	1	1	1
	1,20E+07	32	201	1,4	3	3	6	1	1	1	1
	1,20E+07	64	148	1,7	4	5	9	1	1	1	1
	1,20E+07	128	301	1,8	6	8	26	1	1	1	1
	1,20E+07	256	387	2,4	15	17	64	1	1	1	1
IBM SP2 (switch)	2,60E+07	1	244	1,3	6	6	6	6			6
	2,60E+07	2	1903	7,8	4	5	4	3	6	6	3
	2,60E+07	4	3583	8	3	6	6	2	3	2	2
	2,60E+07	8	5412	11,4	3	9	12	1	2	1	1
Pentium Pro NOW	6,10E+07	1	85	1	2	3	2	2			3
	6,10E+07	2	52745	484,5	9	22	15	5	12	5	7
	6,10E+07	4	139981	1128,5	29	83	74	8	12	6	10
	6,10E+07	8	539159	1994,1	112	357	491	14	18	10	17
	6,10E+07	10	826054	2436,3	174	568	915	18	21	12	20
	6,10E+07	16	288444273	3614,6	28618	170646	454490	1789	1909	205	1791
Digital 8400	2,40E+07	1	28	0,3	6	6	6	6			6
	2,40E+07	2	286	2,5	4	4	4	3	6	6	3
	2,40E+07	3	362	5,3	3	3	3	3	4	3	3
Sun Sparc-5 NOW	7,70E+06	1	82	2,4	20	20	20	20			20
	7,70E+06	2	72966	128,8	72	192	131	41	85	30	45
	7,70E+06	4	369771	266,2	321	1211	1193	85	115	26	89

	7,70E+06	8	835281	322,3	730	3377	5299	95	109	24	97
Multiprocessor Sun Sparc-20	1,00E+07	1	24	0,4	15	15	15	15			15
	1,00E+07	2	54	3,4	8	9	9	8	15	15	8
	1,00E+07	3	74	4,1	6	7	6	6	8	8	6
	1,00E+07	4	118	4,1	5	6	6	5	6	6	5
Hitachi SR2001	5,40E+06	1	31	0,2	28	28	28	28			28
	5,40E+06	2	1165	3	17	20	18	16	30	28	16
	5,40E+06	4	2299	3	12	20	20	9	12	11	9
	5,40E+06	8	3844	3,1	11	29	41	6	7	6	6
	5,40E+06	16	4638	3	12	38	89	4	4	4	4
	5,40E+06	32	6906	4,9	19	66	257	3	3	3	3
Convex Exemplar	1,05E+07	1	60	0,16	15	15	15	15			14
	1,05E+07	2	21373	8,3	20	45	33	14	27	16	14
	1,05E+07	4	64457	9,2	42	153	153	14	18	7	14
	1,05E+07	8	194476	11,3	116	562	895	17	19	5	17
Digital Alpha farm	1,01E+07	1	29	0,3	15	15	15	15			15
	1,01E+07	2	17202	81,1	21	44	32	14	29	18	16
	1,01E+07	3	34356	83	31	88	73	14	21	11	15
	1,01E+07	4	47109	81,3	39	129	125	13	18	9	14
Parsytec GC	1,93E+07	1	98	1	8	8	8	8			8
	1,93E+07	2	6309	113	8	14	11	6	13	10	7
	1,93E+07	4	23538	143	16	43	39	6	8	5	7
	1,93E+07	8	29080	254	34	84	101	6	7	5	6
	1,93E+07	16	224977	342	139	512	1193	10	11	6	10
	1,93E+07	32	130527	658	312	608	1579	11	11	10	11
IBM SP2 (Ethernet)	2,60E+07	1	241	1,3	6	6	6	6			6
	2,60E+07	2	18759	183,6	9	21	15	6	14	8	8
	2,60E+07	4	39025	628,2	29	73	61	9	13	9	12
	2,60E+07	8	88795	1224,1	106	235	260	14	18	14	17

Annexe B

Glossaire.

BSP	Bulk Synchronous Parallelism.
DM	Distributed Memory.
DP	Dynamic Programmation.
DRMA	Direct Remote Memory Access.
INRIA	Institut National de Recherche en Informatique et Automatique.
LIFO	Laboratoire d'Informatique Fondamentale d'Orléans.
LPDA	Logic Push-Down Automata.
MPI	Message Passing Interface.
NUMA	Non-Uniform Memory Access.
P-RAM	Parallel Random Access Machine.
SIMD	Single Instruction Multiple Data.
SM	Shared Memory.
SVM	Shared Virtual Memory.

Table des figures

Bibliographie

[BAMH98] M.Bamha and Gaétan Hains. *A self-balancing join algorithm for SN machines.* To appear in PDCS'98, Las Vegas Octobre 1998, IASTED.

[BOMO72] R. S. Boyer and J. S. Moore. The sharing of structure in theorem-proving programs. In *Machine Intelligence*, chapter 7, pages 101-116. John Wiley and Sons, New York, 1972.

[BSPlib] "BSPlib: The BSP Programming Library" Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B.Rao, Torsten Suel, Thanasis Tsantilas, and Rob Bisseling. Technical report PRG-TR-29-9, Oxford University Computing Laboratory, May 1997. (http://www.bsp-worldwide.org/)

[COLL94] BSP Programming. W.F.McColl, May 1994.

[COLL96-1] W. F. McColl. Universal computing. In L. Bougé, P. Fraigniaud, A. Mignotte, and Y. Robert, editors, *Euro-Par'96, Parallel Processing*, number 1123–1124 in Lecture Notes in Computer Science, Lyon, August 1996. LIP-ENSL, Springer.

[COLL96-2] "Questions and answers about BSP" David B. Skillicorn, Jonathan M. D. Hill and W. F. McColl. Technical Report 15-96, Programming Research Group, Oxford University Computing Laboratory, August 1996.

[GENG95] M.Gengler, S.Ubéda, F.Desprez. *Initiation au parallélisme: concepts, architectures et algorithmes.* Masson 1995.

[GIBB88] A.Gibbons, W.Rytter. *Efficient Parallel Algorithms.*Cambridge University Press 1988.

[GUILL93] M. Dupré. S. Guilloré. Dialect2 et l'analyse syntaxique non déterministe. Rapport de Recherche 93-01, Université d'Orléans, France, 1993.

[HILL97] "BSPlib: The BSP Programming Library" Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B.Rao, Torsten Suel, Thanasis Tsantilas, and Rob Bisseling. Technical report PRG-TR-29-9, Oxford University Computing Laboratory, May 1997. (http://www.bsp-worldwide.org/)

[OXFORD] Oxford Parallel. BSP: the Bulk Synchronous Parallel model. http://www.comlab.ox.ac.uk/oucl/oxpara.html

[PERE85] Fernando C.N. Pereira. *A structure-sharing representation for unification-based grammar formalism.* Inc *Proc. of the 23rd Annual Meeting of the Association for Computational Linguistic*, pages 137-144, 1985.

[SEDG88] Robert Sedgewick. *Algorithms.* Addison-Wesley series in computer science.Addison-Wesley, 2nd ed. edition, 1988.

[VALL90] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, pages 103–111, August 1990.

[VALT97] Rapport de stage de DEA: Indexation de termes logiques au sein d'un évaluateur tabulaire de Programmes logiques. LIFO et INRIA, 1997. L.Valter

[VILL93] E. de la Clergerie. *Automates à Piles et Programmation Dynamique. DyALog: une application à la programmation en Logique.* PhD thesis, Université Paris VII, France, 1993.