Introduction générale

Durant ces dernières années, on assiste à une forte augmentation tant dans le nombre que dans le volume des informations mémorisées par des bases de données scientifiques, économiques, financières, administratives, médicales etc. Le stockage en lui même ne pose pas de réelles difficultés du point de vue informatique, mais le besoin d'interpréter ou de trouver de nouvelles relations entre les éléments stockés dans ces bases a suscité beaucoup d'intérêt. Ainsi, la mise au point de nouvelles techniques informatiques est devenu un thème important pour un bon nombre de chercheurs. Le "**Knowledge Discovery in Databases**" **(KDD)** et le "**Data Mining**" représentent un domaine émergeant essayant de répondre à ces objectifs.

Le Knowledge Discovery in Databases (**KDD**) ou Extraction de Connaissance à partir de Données (**ECD**) est l'extraction d'information implicite, non connue et potentiellement utile, stockée dans des bases volumineuses. Ses concepts s'appuient sur le constat qu'il existe au sein de chaque entreprise des informations cachées dans des gisements de données appelés entrepôt de donnée (Data Warehouse). Ils permettent, grâce à un certain nombre de techniques spécifiques, de faire apparaître des connaissances. Dans la littérature, on distingue différents objectifs du **KDD**, à savoir la classification, le regroupement, la régression, la découverte de règles associatives, etc. En effet, l'information découverte peut être exprimée sous forme d'un ensemble de règles associatives, qui permettent de définir des liens entre les données, et par la suite, prédire la conduite d'autres données différentes de celles stockées dans la base.

Le **Data Mining** ou **Fouille de données** est souvent vu comme un processus équivalent au **KDD**, bien que la plupart des chercheurs voient en lui une étape essentielle de la découverte de connaissance. C'est en effet une étape non triviale du processus d'extraction de connaissance qui consiste à identifier des motifs (patterns) valides, nouveaux, potentiellement utiles et compréhensibles à partir d'une grande collection de données.

Notre mémoire consiste à implémenter, interpréter et comparer deux algorithmes parallèles de Data Mining. Pour cela nous allons dans un premier chapitre définir le Data Mining et présenter ses intérêts et ses domaines d'applications. Nous allons par la suite, dans un second chapitre, illustrer la notion de recherche de règles d'association et étudier différents algorithmes séquentiels et parallèles de Data Mining. A la suite de ce chapitre nous allons présenter les deux algorithmes à implémenter. Cette implémentation sera exposée en détail dans le troisième chapitre accompagnée d'une présentation de l'environnement de travail. Le quatrième chapitre sera, en premier lieu, consacré à l'interprétation des différents mesures et résultats produits par les deux algorithmes et, en deuxième lieu, à une comparaison entre leurs performances: accélération, efficacité et temps de réponse.

Introduction au « DATA MINING »

I. Introduction :

Le terme de Data Mining signifie littéralement "forage de données". Comme dans tout forage, son but est de pouvoir extraire des connaissances. Ainsi nous pouvons définir le Data Mining comme étant l'extraction efficace de l'information non triviale, implicite auparavant inconnu et potentiellement utiles à partir de données. L'accroissement du volume des données stockées dans les bases a fait qu'il est devenu urgent et vital de recourir à des techniques et méthodes pour extraire de l'information à partir de cette masse volumineuse de données. Le Data Mining est alors devenu rapidement un thème de recherche suscitant l'intérêt de la communauté scientifique.

Dans ce chapitre nous allons présenter l'interaction entre le Data Mining et les statistiques, les avantages du Data Mining, ainsi que ses différents domaines d'application.

II. Data Mining et Statistiques: [DMS]

Le Data Mining est le descendant et, selon certains, le successeur des statistiques telles qu'elles sont pratiquées actuellement.

Statistiques et Data Mining ont le même but, qui est de réaliser des «modèles» compacts et compréhensibles rendant compte des relations liant la description d'une situation à un résultat (ou un jugement) concernant cette description. L'hypothèse implicite est bien sûr que le résultat, la mesure, ou le jugement que nous essayons de modéliser dépend effectivement des éléments de description que nous avons.

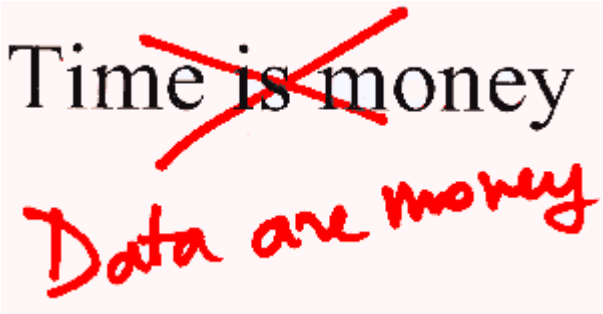
La différence essentielle est que les techniques de Data Mining construisent le dit modèle de manière automatique alors que les techniques statistiques «classiques» requièrent d'être maniées , et guidées par un statisticien professionnel, celui-ci ayant déjà une idée , peut être préconçue, des «hypothèses de dépendance» à formuler.

Les techniques de Data Mining apportent un gain énorme tant en performance qu'en maniabilité ou en temps de travail. La possibilité de réaliser ses propres modèles statistiques par soi-même sans besoin de sous-traiter ou de se concerter avec un statisticien apporte une grande liberté aux utilisateurs opérationnels.

Le Data Mining est la continuation des statistiques par d'autres moyens, plus simples et plus puissants.

On rencontre des «success stories» dans des domaines aussi divers que la gestion de production, les ressources humaines ou la restauration collective.

II.1. Pourquoi le Data Mining est une bonne idée ?



En construisant spontanément un modèle des dépendances au lieu de vérifier les hypothèses d'un statisticien, les techniques de Data Mining ramènent parfois des trésors à la surface, comme par exemple l'association entre le syndrome de Reyes et la prise d'aspirine

chez les enfants, ou, moins sérieusement, la corrélation entre les achats de couches bébé et de lait en poudre dans les supermarchés.

Les techniques de Data Mining nous dispensent d'un statisticien, mais il reste indispensable de maîtriser son métier. Les principaux avantages du data Mining restent la vitesse et la simplicité. Si les conclusions sont relativement de simple bon sens, autant les obtenir en quelques heures plutôt qu'en quelques semaines.

Ces techniques permettent de plus de traiter de grandes quantités d'exemples (plusieurs millions) sans inconvénients. Elles sont également capables de faire face à un grand nombre de variables prédictives (jusqu'à plusieurs milliers).

III. Le Data Mining dans les entreprises: [DME]

III.1. Le contexte économique :

L'accroissement de la concurrence, l'individualisation des consommateurs et la brièveté du cycle de vie des produits obligent les entreprises à non plus simplement réagir au marché mais à l'anticiper. Elles doivent cibler au mieux leur clientèle afin de répondre à ses attentes. La connaissance de son métier, des schémas de comportement de ses clients et de ses fournisseurs est essentielle à la survie de l'entreprise, car elle lui permet d'anticiper sur l'avenir.

III.2. Le stockage des données

Aujourd'hui, les entreprises ont à leur disposition une masse de données importante. En effet, les faibles coûts des machines en terme de stockage et de puissance ont encouragé les sociétés à accumuler toujours plus d'informations. Cependant, alors que la quantité de données à traiter augmente énormément, on estime que la quantité de données collectées dans le monde double tous les 20 mois, le volume d'informations fournies aux utilisateurs n'augmente lui que très peu. Ces réservoirs de connaissance doivent être explorés afin d'en comprendre le sens et de déceler les relations entre données et les modèles expliquant leur comportement.

III.3 Les domaines d'application :

Parmi les domaines concernés par le Data Mining et le Data Warehouse, c'est probablement celui de la fidélisation de clientèle qui mobilisera le plus d'attention dans les prochaines années ; en effet, il est aujourd'hui surprenant de voir avec quelle aisance, un bon nombre d'entreprises évoquent le montant des ventes pour chacun de leurs produits, alors qu'en même temps, elles sont le plus souvent incapables d'identifier les mêmes informations pour leurs principaux clients. Plus grave encore, un grand nombre d'entreprises sont incapables aujourd'hui de connaître des informations aussi basiques que le nombre de leurs clients.

Dans l'ère industrielle où il est souvent impossible d'associer à chaque client un "chargé de compte", la mémoire humaine de ceux-ci laisse désormais la place à une mémoire d'entreprise, capable de mémoriser les habitudes de consommation de chacun de ses clients, son profil, son potentiel d'achat, ou encore ses demandes fréquentes.

Les exemples suivants ne sont qu'une petite illustration des domaines où le Data Mining peut jouer un rôle important.

- **Analyse de risque**

  Les compagnies d'assurance peuvent rechercher les caractéristiques de clients à haut risque, les compagnies bancaires, déterminer si un crédit peut ou non être accordé à une certaine personne.

- **Marketing direct**

  Déterminer les caractéristiques (âge, sexe, profession, habitation, région, ...) de la population à cibler pour un publipostage. Le courrier pourra ainsi être envoyé à la population offrant la plus forte probabilité de réponse.

- **Grande distribution**

  Déterminer les profils des consommateurs, leur modèle d'achat, l'effet des périodes de soldes ou de publicité, déterminer le « panier de la ménagère ». Piloter le processus par la demande (le produit souhaité) plutôt que par l'offre (le produit poussé). Cela permet d'améliorer les ventes en présentant le bon produit au bon endroit et au bon moment, de réduire les coûts logistiques par rationalisation des références et des stocks de produits obsolètes, facilite la négociation des achats.

- **Gestion des stocks**

  Savoir quand commander un produit, quelle quantité demander...

- **Maintenance**

  Pour le support technique, déterminer en fonction des pannes rencontrées, les causes des problèmes et la manière de les résoudre.

- **Contrôle de qualité**

  Déterminer quels sont les facteurs de production qui contribuent à la qualité des produits, quelles sont les combinaisons de facteurs (machine, individu, ...) qui est la source de produits défectueux.

- **Domaine médical**

  Plusieurs types d'utilisation peuvent être faites dans le domaine médical :

  - Diagnostic, découvrir la maladie d'après les symptômes du patient.
  - Choix du médicament le plus approprié pour guérir une maladie donnée, pour un individu donné.
  - Recherche scientifique.

- **Analyse financière**

  Maximiser le retour sur investissement de portefeuilles d'actions.

- **Télécommunications**

  Identifier les clients susceptibles de résilier leur abonnement et d'en révéler les principaux motifs. A l'heure où chaque nouveau client "s'achète" de plus en plus cher, la nécessitée d'un système d'information efficient pour éviter d'en perdre est évident.

- **Industrie**

  Analyser la répartition des ventes par produits, par ensemble de produits, par clients ou par marché, y compris à une échelle internationale.

- **Administration**

  Détecter les fraudes pour identifier les tricheurs potentiels.

IV. Conclusion :

L'arrivée du Data Mining est seulement la dernière étape de l'introduction de méthodes quantitatives, scientifiques dans le monde du commerce, de l'industrie et des affaires. Maintenant tous les non statisticiens, c'est-à-dire 99,5 % d'entre nous, peuvent construire des modèles exacts de certaines de leurs activités, pour mieux les étudier, les comprendre et les améliorer.

Pour la première fois de leur histoire, les statistiques sortent des mains des spécialistes. L'art du spécialiste est remplacé par des méthodes nouvelles qui donnent des résultats aussi bons ou meilleurs sans demander de connaissances spécialisées.

Le Data Mining est certainement parmi les applications les plus utiles de la puissance croissante des ordinateurs, et les domaines de recherche les plus intéressants de l'Informatique Avancée [DMS].

Les Algorithmes de « DATA MINING »

I. Introduction :

Aprés le premier chapitre où nous avons essayé d'expliquer le Data Mining et ses domaines d'application nous allons dans ce chapitre exposer plus ou moins en détail quelques uns des algorithmes de Data Mining. Nous nous intéresserons particulièrement aux algorithmes de génération des règles d'associations. Pour cela, nous allons tout d'abord introduire la notion de règles d'associations et expliquer la méthode de recherche de ces règles proposées par **Agrawal** afin de mieux comprendre le fonctionnement des algorithmes séquentiels et parallèles de Data Mining.

II. Règles d'association :

Une règle associative est de la forme A → B, où A et B sont des conjonctions d'attributs de la base de données. « A » représente l'antécédent (ou la prémisse) de la règle et « B » la conclusion. Par exemple, si on dispose d'une base de données contenant des informations sur la population de Tunis, la règle associative suivante peut être extraite:

$$(\text{Age}>25) \wedge (\text{Revenu}<8000) \rightarrow (\text{Modèle\_voiture} = \text{populaire})$$

Cette règle associe l'age et le revenu d'un individu au type de la voiture qu'il peut posséder en Tunisie.

L'association d'un processus de découverte d'associations avec un système guidé par l'utilisateur, devient intéressant dans le contexte du Data Mining. En effet, l'utilisateur peut intervenir pour mettre un système de filtrage pour écarter les associations non évidentes ou ayant peu d'intérêt. A cet effet, les paramètres les plus couramment utilisés, dans un système de filtrage, sont le « support » (valeur qui permet de mesurer la fréquence de l'association) et la « confidence » (valeur permettant de mesurer la force de l'association) d'une règle.

II.1. Recherche de règles d'association :

Le problème d'extraction de règles d'association a été introduit la première fois par **Agrawal**. Il fut développé à l'origine pour l'analyse de bases de donnés de transactions de ventes. Chaque transaction est constituée d'une liste de produits achetés, afin d'identifier les groupes d'articles les plus vendus fréquemment ensembles.

Exemple de règles d'association : « Un client qui achète du chocolat et du lait a tendance à acheter du fromage »

La principale application est donc « l'analyse du panier de la ménagère », mais aujourd'hui, cette technique est appliquée à tout domaine cherchant à regrouper des objets entre eux à partir de grandes bases de données.

II.2. Concepts de base : [RA, TI & AS]

Soit **I = {i1, i2,…, im}** un ensemble d'objets ou d'items. Soit **D** un ensemble de transactions tel que chaque transaction **T** soit un ensemble d'objets vérifiant **T ⊂ I**. A chaque transaction **T**, on associe un unique entier, faisant fonction d'identifiant, appelé son **TID**. On dit qu'une transaction **T** contient **X**, un ensemble d'objets quelconques de **I**, **si X ⊂ T**.

Une règle d'association est une implication de la forme :

$\mathbf{X \Rightarrow Y}$, avec $\mathbf{X \subset I}$, $\mathbf{Y \subset I}$ et $\mathbf{X \cap Y = \varnothing}$.

Où **X** est appelé antécédent, prémisse ou partie gauche de la règle et **Y** est appelé conséquent, conclusion ou partie droite de la règle.

La règle X $\Rightarrow$ Y a un **support** « **s** » dans l'ensemble des transactions **D** si **s%** des transactions de D contiennent $\mathbf{X \cup Y}$.

$$\mathbf{Sup(X \Rightarrow Y) = Sup(X \cup Y) = P(X, Y) = P(X \cup Y) / |D|}$$

La règle $\mathbf{X \Rightarrow Y}$ se reporte à l'ensemble de transactions avec une **confiance** (*confidence*) « **c** » si **c%** des transactions **D** qui contiennent l'ensemble des objets **X** contiennent aussi l'ensemble des objets **Y**.

$$\mathbf{Conf(X \Rightarrow Y) = Sup(X \cup Y) / Sup(X)}$$

La confiance peut aussi se définir en termes de probabilité conditionnelle

$$\mathbf{Conf(X \Rightarrow Y) = P(Y|X) = P(X \cap Y) / P(X)}$$

Avec : **P(X)** le nombre d'apparition de **X** dans **D**.

| D | le nombre de transactions de la base **D**.

La complexité de génération des règles est $\boldsymbol{O}\mathbf{(r\text{-}2L)}$, où **r** est le nombre d'items fréquents et L la longueur de l'item le plus long.

Etant donné un ensemble de transactions, le problème de la recherche de règles d'association consiste à générer toutes les règles qui ont un support et une confiance supérieurs à un seuil spécifié par l'utilisateur (*minsup* et *minconf*).

Vue l'énorme intérêt pratique que présente ce problème dans le processus d'extraction de connaissance dans les bases de données plusieurs équipes de recherche ont développé plusieurs algorithmes mais tous ces algorithmes se basent sur la même idée introduite par **Agrawal**. Une règle n'a de réelle utilité que si elle satisfait un certain taux minimal de fréquence *minsup* et un taux de confiance *minconf* fixés par l'utilisateur. Dans le cas contraire toute règle est à rejeter. **Agrawal** part donc du faite que pour que le support d'une règle satisfasse *minsup*, la prémisse et la conclusion de cette règle doivent impérativement satisfaire *minsup*. Le problème de recherche de règles d'association peut par conséquent être décomposé en deux sous problèmes :

1. Trouver tous les ensembles d'objets (*itemsets*) qui ont un support supérieur au support minimum *minsup*. Le support pour un ensemble d'objets correspond au nombre de

transactions qui contiennent cet ensemble d'objets. Les itemsets qui atteignent le support minimum sont appelés les ensembles d'objets fréquents (*itemsets fréquents*).

2. Utiliser ces ensembles d'objets fréquents (*itemset fréquent*) pour déduire les règles recherchées. Ainsi, si ABCD et AB sont des ensembles d'objets fréquents, alors on peut générer la règle AB $\Rightarrow$ CD. Pour savoir si cette règle est à retenir ou à rejeter, on calcule la confidence de AB $\Rightarrow$ CD qui n'est autre que le ratio : conf = sup (ABCD) / sup (AB).

Si conf $\geq$ minconf, alors la règle est retenue (la règle aura forcément un support minimum car ABCD est fréquent).

On note que La première étape est la plus coûteuse en temps d'exécution, c'est pourquoi notre intérêt c'est porté essentiellement sur cette dernière. C'est à dire sur le problème de génération des *itemsets fréquents*.

III. Les algorithmes séquentiels :

Dans la littérature, on distingue deux principaux types d'algorithmes : une première génération séquentiel et une deuxième qui vise à paralléliser le traitement afin de réduire les temps de réponse.

III.1. Apriori : [RA & JS]

Apriori, proposé par **Rakesh Agrawal** et **John Shafer** en février 1996 est l'algorithme fondamental. C'est en faite une amélioration d'une idée dont le principe général est le suivant :

1. Au début, les supports des *itemsets* de taille 1 (noté : *1-itemsets*) sont calculés en effectuant une première passe sur la base de données.

2. Les *itemsets* dont le support n'a pas atteint le seuil *minsup*, défini par l'utilisateur, sont écartés. A ce niveau, l'ensemble obtenu est appelé ensemble des *items* fréquents de taille 1 (noté : *1-itemset_freqents*). Ce dernier servira à générer un nouvel ensemble des *items* de taille 2 dit ensemble des *items* candidats (noté : *2-itemset_candidats*). Au cours de l'étape suivante les supports de ces *2-itemset_candidats* sont alors calculés et après élimination des non fréquents un

ensemble dit des *3-itemset_candidats* est généré et ainsi de suite jusqu'à ce que on ne peut plus générer de candidats.

III.1.1. L'algorithme :

k-itemset: un *itemset* qui contient k items.

L_k : ensemble des *k-itemset_fréquents*.

C_k: ensemble des *k-itemset_candidats*.

D: la base de données.

L_1:= {fréquent *1-itemset*};
k := 2; /* k représente le nombre de passes*/
While (**$L_{k-1} \neq \emptyset$**) **do**
Begin
/*générer les items candidats*/
C_k := nouveaux candidats de taille k génères de l'ensemble L_{k-1};
For all transactions **T∈ D do**
Incrémenter le nombre de candidats dans **C_k** contenu dans la transaction **T**;
/*génération des items fréquents*/
L_k: = tous les candidates dans **C_k** avec un support minimum;
k := **k** + 1;
End.

L'algorithme Apriori

La première boucle d'Apriori consiste simplement à compter les occurrences de chaque objet afin de définir les *1-itemsets_fréquents*. Ensuite, chaque boucle se décompose en deux phases. D'abord, les ensembles fréquents **L_{k-1}** formés à la **$(k-1)^{emme}$** itération sont utilisés pour former l'ensemble candidat **C_k**, à l'aide de la fonction de génération des candidats décrite ci-dessous. Puis, la base de données est scannée pour pouvoir déterminer le support de chaque candidat. L'ensemble des *k-itemset_frequent* est ensuite généré en éliminant les *items* dont le support n'a pas atteint le seuil minimum *minsup*.

- **Génération des candidats :**

La fonction de génération des candidats prend en argument $\mathbf{L_{k-1}}$, l'ensemble des *(k-1)-itemset_fréquents*, et retourne l'ensemble des *k-itemset_candidats*. On peut diviser cette fonction en deux étapes : la première (join step) permet de former un ensemble de $\mathbf{C_k'}$ à partir de L_{k-1}, la deuxième (prune step) permet d'obtenir l'ensemble $\mathbf{C_k}$ tel qu'il a été défini auparavant : pour cela, on élimine tous les candidats « **c** » de $\mathbf{C_k}$ dont au moins un sous-ensemble n'est pas dans $\mathbf{L_{k-1}}$ (sous-ensemble qui n'est pas fréquent). Cet élagage se fait en se basant sur la propriété des motifs fréquents : un motif non fréquent ne peut pas être le sous-ensemble d'un motif fréquent. Cette fonction permet de générer un nombre minimal de candidats.

Première étape (Join step) :

Soit $\{X = a_1,a_2,\ldots,a_k\}$ et $\{Y = b_1,b_2,\ldots,b_k\}$ deux items fréquents de taille k.

Si $a_i = b_i$ pour tout $1 \leq i \leq k-1$

Alors un item candidat $\{Z = a_1, a_2, \ldots a_{k-1}, a_k, b_k\}$ de taille k+1 généré.

Deuxième étape (Prun step) :

L'item candidat Z ne sera gardé dans C_k que si tous les sous items W de taille k formés à partir de Z soient fréquents.

III.1.2. Exemple :

minsup = 2.

Tid	Items
1	A C D E
2	AB C
3	A B C
4	ABE
5	BE
6	BC

Tableau 1 : La base de données

L'ensemble des candidats de taille 1 :C_1= {A, B, C, D, E}

Premier itération :

Calcul du support des *1-itemset_candidats*.

Items	A	B	C	D	E
Support	4	5	4	1	3

Tableau 2 : *1-itemset_candidat*

Génération de l'ensemble L_1= {A, B, C, E} des *1-itemset-frequent* en éliminant les items non fréquents.

Items	A	B	C	E
support	4	5	4	3

Tableau 3 : *1-itemset_frequent*

Deuxième itération :

Génération de C_2= {AB, AC, AE, BC, BE, CE} ensemble des *2-itemset_candidats* puis calcul des supports.

Items	AB	AC	AE	BC	BE	CE
support	3	3	2	3	2	1

Tableau 4 : *2-itemset_candidat*

Génération de l'ensemble L_2= {AB, AC, AE, BC, BE,} des *1-itemsets-frequent*.

Items	AB	AC	AE	BC	BE
support	3	3	2	3	2

Tableau 5 : *2-itemset_frequent*

Troisième itération :

Génération de C_3= {ABC, ABE, ACE, BCE}, élimination des items ACE et BCE car CE n'est pas fréquent, l'ensemble des *2-itemset_candidats* devient donc C_3= {ABC, ABE} puis calcul des supports.

Items	ABC	ABE
support	2	1

Tableau 6 : *3-itemset_candidat*

Génération de l'ensemble L_3 = {ABC}

Quatrième itération :

C_4 =∅, aucun candidat ne peut être généré.

Fin de l'algorithme

III.2. AprioriTID : [RA & RS]

Ce second algorithme AprioriTID est quasiment identique à Apriori sauf qu'il présente une caractéristique qui le distingue. En effet dans AprioriTID, dés la seconde passe, la base de données n'est plus utilisée pour calculer les supports. On utilise plutôt un ensemble intermédiaire, noté $\mathbf{C_k'}$, déduis à partir de l'ensemble des items candidats génères lors de la passe précédente.

Cette caractéristique constitue en fait une amélioration de l'algorithme Apriori, car lors des dernières passes, la taille de $\mathbf{C_k'}$ peut devenir beaucoup plus réduite que celle de la base de données **D**. Ainsi nous économisons au niveau des entrées/sorties nécessaires pour lire la base.

III.2.1. L'algorithme :

k_itemset: un *itemset* qui contient k items.

L_k : ensemble des *k-itemset_fréquents*.

C_k: ensemble des *k-itemset_candidats*.

C_k': L'ensemble intermédiaire qui va remplacer la base de données D.

D: la base de données.

L_1 = {1-itemset_frequent};

C_1' = {D} /* La totalité de la base de données */

k := 2; /* k représente le nombre de passes */

While ($L_k \neq \emptyset$) **Do**

Begin

/* générer les items candidats* /

C_k := nouveaux candidats de taille k génères de l'ensemble L_{k-1};

Ck' = ∅ ;

For all transactions T∈ D **Do**

/* déterminer les items candidats ∈ à C_k et figurant dans T */

C_t **: =** {c ∈ C_k | (c-c[k]) ∈ T ∧ (c-c[k-1]) ∈ T };

For all c ∈ C_t Do

c.count ++;

Si ($C_t \neq \emptyset$) **Alors** $C_k' = C_k' \cup \{T.TID, C_t\}$;

End Do

/* génération des items frequents */

L_k = {**c ∈ C_k** | c.count ⟨ *minsup*};

k := **k** + 1;

End

Réponse: = $\cup_k L_k$

L'algorithme AprioriTID

IV. Les algorithmes parallèles :

Suite à l'expansion des supports physiques de stockage et les besoins incessant de sauvegarder de plus en plus des données, les algorithmes séquentiels de recherche des règles d'associations se sont avérés inefficace. Ainsi l'introduction des nouvelles versions parallèles est devenue impérative. Parmi ces algorithmes parallèles nous avons choisi de voir en détail les trois familles : Count Distribution, Data Distribution et Intelligent Data Distribution.

IV.1. L'algorithme Count Distribution (CD) : [RA & JS]

Dans l'algorithme **CD,** proposé par **R.Agrawal** et **J.C.Shafer,** chaque processeur calcule le nombre d'apparition de tous les candidats dans la portion de la base de données de transactions qui est stockée dans sa mémoire centrale. Par la suite une seule passe est effectuée sur la portion locale de la base pour calculer le nombre d'apparition partielle. Le nombre d'apparition totale des candidats est par la suite calculé par une opération de réduction globale, ce traitement est donc analogue à une exécution parallèle de l'algorithme **Apriori** par l'ensemble des processeurs chacun sur sa portion locale de la base de données (voir figure1). Les processeurs sont donc indépendants et il n'y a communication entre eux qu'à la fin du calcul. Cependant cet algorithme ne divise pas l'ensemble des candidats ce qui constitue une redondance de calcul. De plus dans le cas où le nombre de candidats est très élevé, la structure de données utilisée pour stocker l'ensemble des candidats sera par conséquent assez volumineuse et ne pourra pas être chargé en totalité dans la mémoire centrale, chaque processeur sera donc obligé de la partitionner en plusieurs parties et à chaque fois effectuer une passe correspondant à la portion qui est chargée en mémoire et effectuer par la suite une opération d'entrée/sortie pour charger une autre portion et ainsi de suite. Le temps de calcul sera alors alourdi par un temps d'entrée/sortie assez important ce qui dégrade considérablement les performances de cet algorithme qui donne de très bons résultats sur les petites bases de données où le nombre de candidats n'est pas très élevé d'autant plus qu'il n'y a presque pas de communication entre processeurs. Notons que le nombre de candidats augmente dans les cas où le nombre d'items distincts dans la base est élevé et/ou que le support minimal *minsup* est très bas. Cet algorithme (**CD**) est donc très efficace pour des bases de données dont le nombre d'items distincts est réduit, s'il est exécuté avec un support minimal assez élevé.

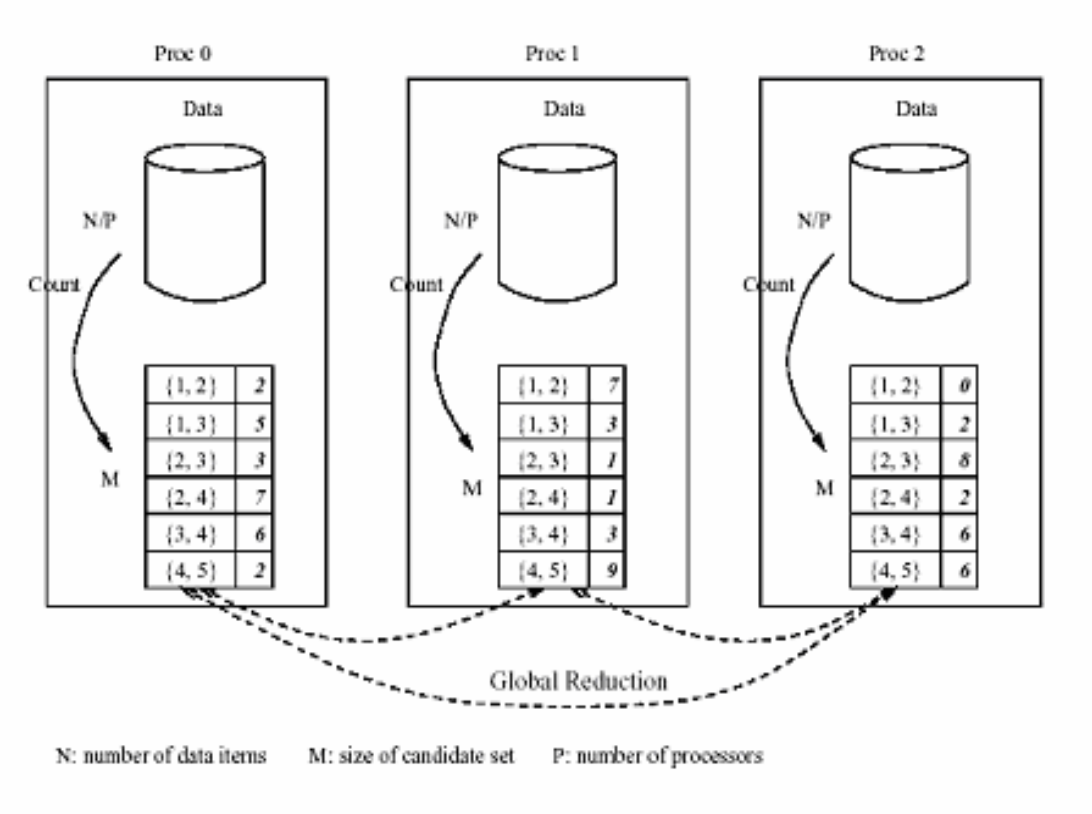


Figure 1 : Count Distribution (CD)

IV.1.1. Les étapes de l'algorithme Count Distribution :

$\mathbf{C_k}$: *k_itemsets_condidats*.
$\mathbf{L_k}$: *k_itemsets_frequents*.
$\mathbf{C_k^i}$: *k_itemsets_condidats* du processeur P_i.
$\mathbf{L_k^i}$: *k_itemsets_frequents* du processeur P_i.
$\mathbf{D_i}$: la portion de la base de transaction qui est stockée dans la mémoire centrale du processeur $\mathbf{P_i}$.
N : nombre de processeurs.

Cet algorithme, comme nous l'avons déjà vu, utilise le principe de redondance de traitement pour éviter les communications entre processeurs .il est décrit par les étapes situés ci-dessous :

❖ **Etape préliminaire :**

Au cours de cette étape on génère l'ensemble des items fréquents de taille 1 (*1-itemset_frequents).*

On distingue trois sous étapes exécutées par tous les processeurs :

1. Parcourir la portion locale de la base de données et déterminer les différents items qui y figurent avec leur nombre d'apparition.

2. Transmettre le résultat aux autres processeurs.

3. Faire la mise à jour du résultat localement calculé.

A la fin, chaque processeur possède donc exactement le même ensemble des *1-itemset_frequents.*

Cette étape est exécutée une seule fois, c'est une phase d'initialisation de l'algorithme.

❖ **Première étape :**

Chaque processeur génère l'ensemble des *k_itemset_condidats* $\mathbf{C_k}$ à partir de l'ensemble des *(k-1)-itemset_frequents* $\mathbf{L_{k-1}}$ calculé à la dernière étape de la passe précédente. On note que, tous les processeurs génèrent le même ensemble $\mathbf{C_k}$ puisque ils possèdent tous le même $\mathbf{L_{k-1}}$.

❖ **Deuxième étape :**

Chaque processeur P_i parcoure sa portion locale de base de données et calcule le support des items candidats génères dans la première étape.

❖ **Troisième étape :**

Echange des supports calculés à l'étape précédente entre les différents processeurs.

❖ **Quatrième étape :**

Chaque processeur génère l'ensemble $\mathbf{L_k}$ à partir de $\mathbf{C_k}$.

❖ **Cinquième étape :**

Chaque processeur, indépendamment des autres, décide de terminer ou continuer vers la prochaine passe. On note que les décisions seront identiques puisque ils ont le même $\mathbf{L_k}$.

Remarque : La sauvegarde des *1-itemsets_frequents se* fait dans un ordre croissant afin d'optimiser le traitement au cour des étapes suivantes.

IV.2. L'algorithme Data Distribution (DD) : [RA & JS]

Dans l'algorithme **DD R.Agrawal** et **J.C.Shafer** ont traité le problème de mémoire de l'algorithme **CD** en partitionnant l'ensemble des candidats entre les processeurs. Ce partitionnement est aléatoire, en effet l'algorithme **DD** utilise le mode **Round Robin. DD** partitionne donc et la base de données et l'ensemble des candidats. Chaque processeur calcule alors les nombres d'apparition des candidats, qui lui sont attribués, dans la totalité de la base pour cela il a besoin d'accéder aux portions de la base qui sont stockées chez les autres processeurs aussi bien que sa portion locale (voir figure2). Cela est effectué de la manière suivante :

Chaque processeur alloue **P** buffers (de taille une page, un buffer pour chaque processeur). Pour le processeur $\mathbf{P_i}$ le i^{emme} buffer est utilisé pour stocker les transactions de la portion locale de la base de données, les autres sont utilisés pour stocker les transactions provenant des autres processeurs. Chaque processeur $\mathbf{P_i}$ contrôle les **P** buffers pour voir lequel contient de l'information. Soit ce buffer **j,** le processeur calcule le nombre d'apparition de ses candidats dans cette page et met à jour le compte ultérieur. Maintenant si ce buffer correspond à celui qui stocke la portion locale (c à d que $\mathbf{i = j}$), il est alors envoyé à tout les autres processeurs (via une transmission asynchrone). Si ce buffer correspond à un buffer qui stocke une transaction provenant d'un autre processeur (c à d que $\mathbf{i \neq j}$) alors son contenu est effacé et il est marqué prêt pour une autre opération de réception asynchrone provenant d'un autre processeur. Ces opérations se déroulent jusqu'à ce que tous les processeurs aient accédé à toutes les transactions de la base. A ce stade tout les processeurs ont calculé le nombre total d'apparition de leurs candidats locaux respectifs, chacun détermine alors l'ensemble des items fréquents à partir de l'ensemble des candidats. Cet ensemble est alors transmis à tout les autres processeurs par une opération de diffusion générale (**all-to-all broadcast**).

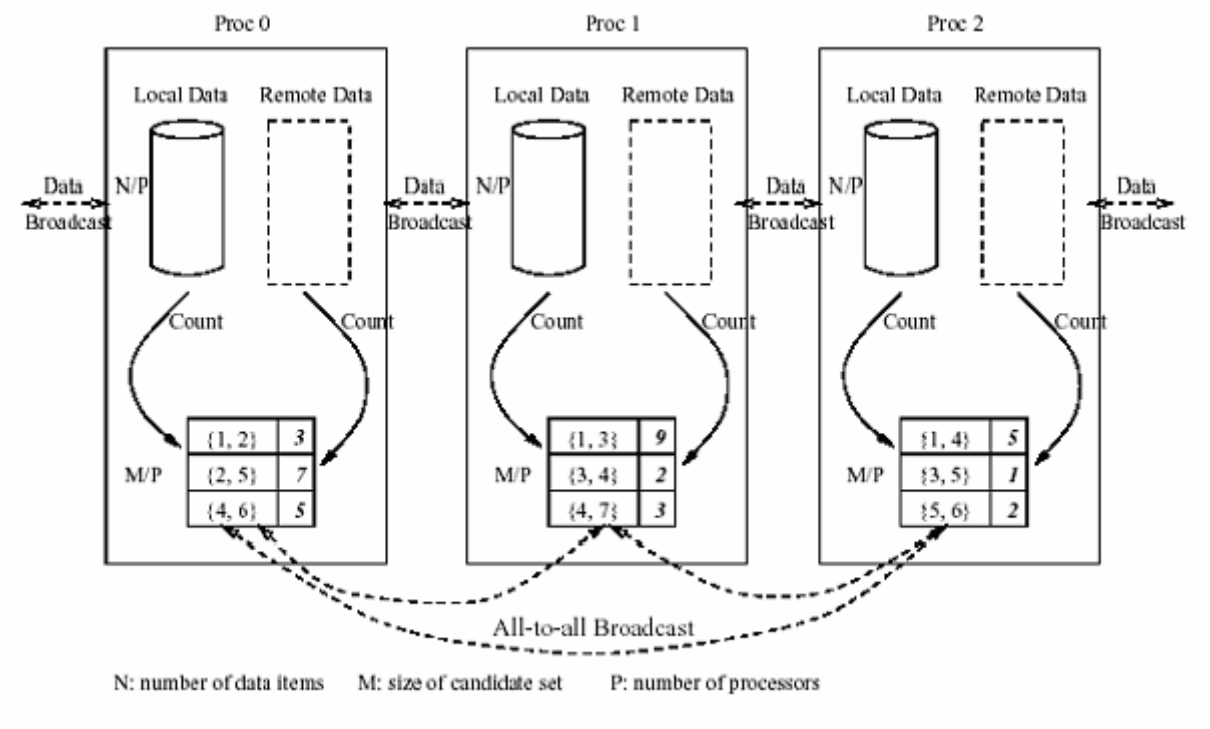


Figure 2 : Data Distribution (DD)

IV.2.1 Les étapes de l'algorithme Data Distribution :

$\mathbf{C_k}$: *k_itemsets_condidats*.
$\mathbf{L_k}$: *k_itemsets_frequents*.
$\mathbf{C_k^i}$: *k_itemset_condidats* du processeur P_i.
$\mathbf{L_k^i}$: *k_itemset_frequents* du processeur P_i.
$\mathbf{D_i}$: la portion de la base de transaction qui est stockée dans la mémoire centrale du processeur $\mathbf{P_i}$.
N : nombre de processeurs.

❖ **Etape préliminaire :**

Cette étape est identique à celle décrite dans l'algorithme CD.

❖ **Première étape :**

Le processeur $\mathbf{P_i}$ génère $\mathbf{C_k}$ à partir de $\mathbf{L_{k-1}}$, ne retient que 1/**N** de l'ensemble précédemment formé (la partie en question est choisie grâce a l'identificateur de processeur) cette partie est noté $\mathbf{C_k^i}$ et elle sera unique.

❖ **Deuxième étape :**

Le processeur $\mathbf{P_i}$ calcule le support de chaque élément de l'ensemble $\mathbf{C_k^i}$ à partir de sa base locale et des bases stockées chez les autres processeurs (cette étape est la plus importante et la plus coûteuse, elle sera un peut plus détaillée ci-dessous).

❖ **Troisième étape :**

Le processeur $\mathbf{P_i}$ détermine $\mathbf{L_k^i}$ et cela en éliminant les items dont le support n'a pas atteint le minimum *minsup*.

❖ **Quatrième étape :**

Le processeur $\mathbf{P_i}$ transmet $\mathbf{L_k^i}$ à tous les autres processeurs. Chaque processeur dispose maintenant de $\mathbf{L_k}$ qui servira à construire $\mathbf{C_k}$ à l'étape suivante.

On remarque que la deuxième étape de l'algorithme DD est la plus importante, en effet elle comporte les principaux traitements : accès à la base, calcul de support et communication.

Cette étape est aussi la plus coûteuse en temps d'exécution.

Détail de la deuxième étape :

Le processeur $\mathbf{P_i}$

1. alloue **N** buffers **B** (**N** : nombre de processeur)

2. charge une page de $\mathbf{D_i}$ dans le buffer $\mathbf{B_i}$

3. effectuer un broad cast de $\mathbf{B_i}$

4. boucler jusqu'à ce que tous les buffers soient traités :

si buffer non_traité et buffer non_vide

alors traiter le buffer

sinon passer au buffer suivant

fin si

Fin de la boucle

5. Barrière de synchronisation.

6. si la base D_i n'est pas traitée en totalité alors reprendre en 3.

Fin.

Traiter un buffer B_j revient à :

Calculer le support des candidats dans ce buffer

Faire une mise à jours du compte ultérieur

Si i = j alors charger une nouvelle page de D_i.

Si i ≠ j alors vider le buffer.

IV.3. L'algorithme Intelligent Data Distribution (IDD) : [RA & JS]

L'algorithme IDD a était développé pour remédier aux problèmes de temps de communication élevé de l'algorithme DD provoqué par des opérations d'émission et de réception collectives de buffers assez volumineux (voir figure3).

Dans IDD, on ne fait plus de all-to-all broadcast. En effet les processeurs sont vues comme étant un anneau logique, où chacun connaît son prédécesseur et son successeur et possède un buffer d'émission (**SBuf**) et un buffer de réception (**RBuf**).

Le processus d'échange de données entre les processeurs se fait comme suit :
Initialement **SBuf** contient un bloc de la base de données, ensuite chaque processeur entame une opération d'envoie asynchrone de **SBuf** à son successeur et une opération de réception asynchrone dans **RBuf** en provenance de son prédécesseur.

Pendant que ces opérations d'émissions et de réceptions se déroulent, chaque processeur parcoure les transactions de **SBuf** et met à jour le support des items candidats locaux. Après l'opération de mise à jour, le processeur attend la fin des communications asynchrones, celle-ci étant terminé, les contenus de **SBuf** et **RBuf** sont permutés. Ainsi l'opération précédente se poursuit (**P**-1) fois ; avec **P** : le nombre de processeurs.

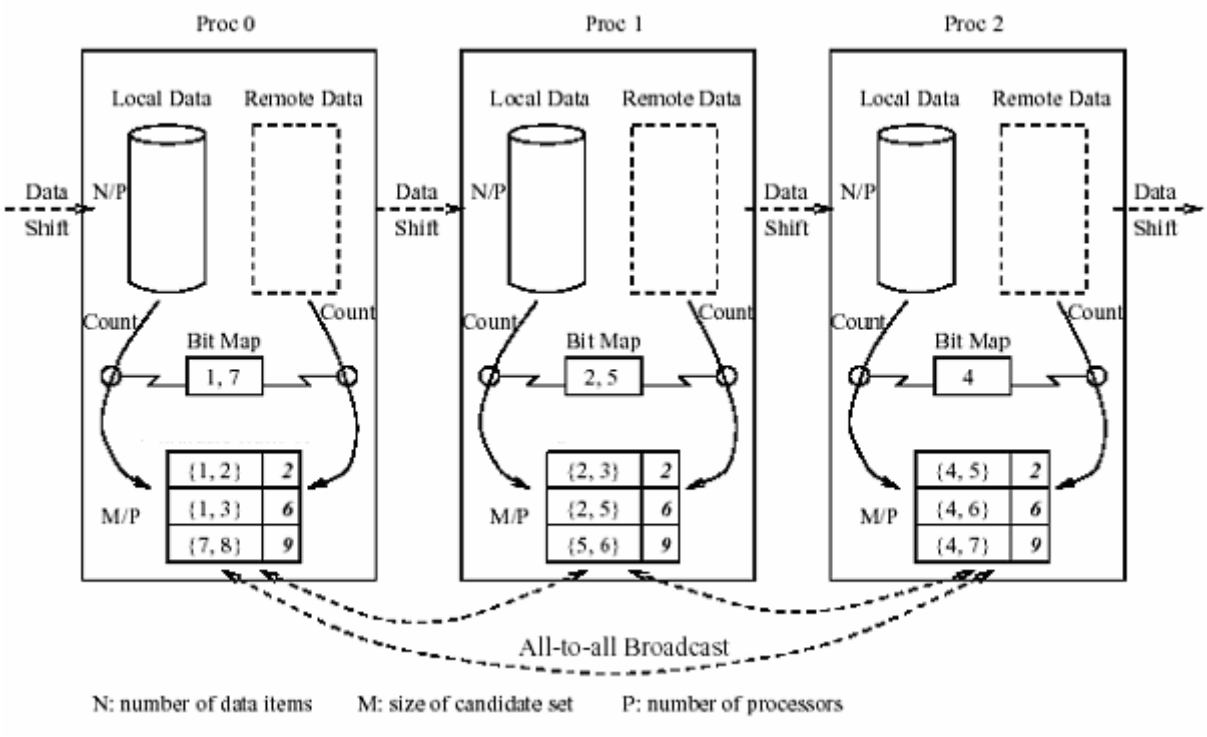


Figure 3 : Intelligent Data Distribution (IDD)

IV.3.1. Les étapes de l'algorithme Intelligent Data Distribution :

La différence entre cet algorithme et l'algorithme DD, vu précédemment, se manifeste dans la deuxième étape.

BSend : Buffer d'émission.

BRecv : Buffer de réception.

$\mathbf{D_i}$: la portion de la base de transaction qui est stockée dans la mémoire centrale du processeur $\mathbf{P_i}$.

N : nombre de processeurs.

Détail de la deuxième étape :

Le processeur $\mathbf{P_i}$

1. alloue deux buffers **BSend** et **BRecv.**

2. charge une page de $\mathbf{D_i}$ dans le buffer **BSend.**

3. Répéter le traitement suivant **N-1** fois :

Déclencher une opération d'émission asynchrone de **BSend.**

Déclencher une opération de réception asynchrone dans **BRecv.**

Traiter **BSend.**

Attendre la terminaison des deux opérations de transmissions asynchrones.

Copier le contenu de **BRecv** dans **BSend.**

4. si la base $\mathbf{D_i}$ n'est pas traitée en totalité alors reprendre en **2.**

Fin.

Traiter le buffer **BSend** revient à :

Calculer le support des candidats dans ce buffer

Faire une mise à jour du compte ultérieur

V. Conclusion :

Suite à une étude approfondie des algorithmes de recherches de règles d'associations présentées ci-dessus, nous avons choisi d'implémenter les deux familles d'algorithmes DD et IDD qui vont nous permettre de toucher de plus prêt les spécificités de la communication inter-nœud de notre machine parallèle.

Implémentation des algorithmes DD et IDD

I. Introduction :

Nous avons dans le chapitre précèdent illustré la notion de recherche de règles d'associations et présenté l'algorithme fondamental de Data Mining *Apriori*, une variante de cet algorithme *AprioriTID* et les trois versions parallèles d'*Apriori* CD, DD et IDD. Nous allons maintenant dans ce chapitre exposer notre implémentation des deux algorithmes parallèles DD et IDD en respectant les spécificités de l'environnement de travail : la machine SP2 de IBM et la bibliothèque d'échanges de messages pour machines parallèles homogènes MPI (Message Passing Interface).

II. Environnement de travail :

II.1. Vue globale :

L'acronyme SP2 signifie Scalable Power parallel Systems second generation. La machine SP2 est composée d'une armoire (frame) pouvant contenir jusqu'à 16 machines connectées entre elles avec des liens rapides. Nous disposons actuellement de 8 serveurs (noeuds) dans cette armoire. Cette machine est contrôlée par un serveur dédié. [IBM1] Schématiquement la SP2 se présente de la façon suivante :

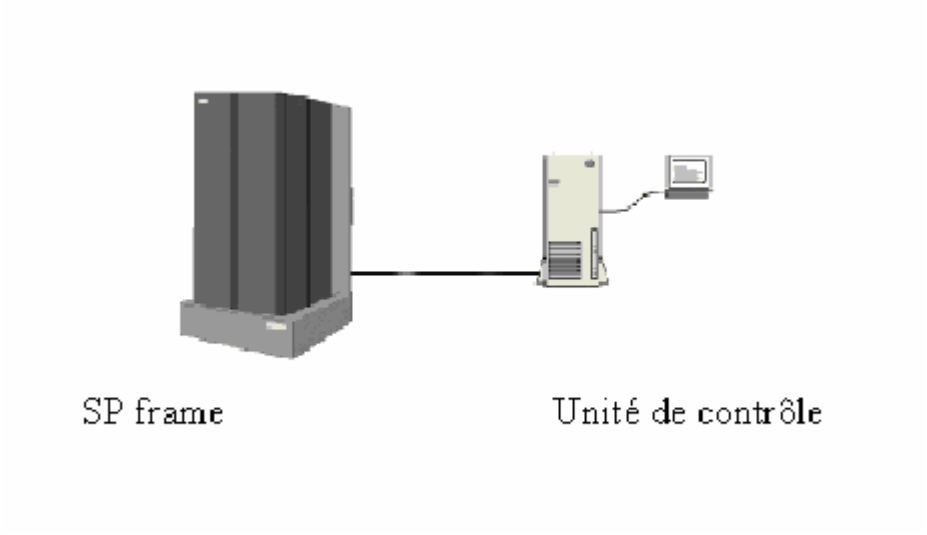


Figure 4 : Architecture de la SP2

II.2. Caractéristiques de la SP2 : [IBM2] [Jussieu]

- Type : noeud fin SMP.
- Nombre : 8 noeuds (extensible jusqu'à 128 noeuds).

Processeurs :

- PowerPc 604e à 375 Mhz.
- Cache L1 : 32 Ko pour les instructions et 32 Ko pour les données.
- Cache L2 : 4 Mo.
- Nombre : 4 processeurs.
- Puissance : 0, 664 Gflop/s.
- Puissance de crête : 21, 248 Gflop/s.

Architecture du Bus Système :

- Opère à 100 Mhz.
- 2 emplacements d'extension PCI 32 bits.

Mémoire :

- 128 Mo de SDRAM synchrone ECC (Error Checking and Correcting) à 100 MHz.
- Extensible jusqu'à 16 Go.
- Largeur du Bus Mémoire : 128 bits.

Entrées/Sorties :

- Carte mère intègre directement un contrôleur Ethernet 10/100 Mbits et un adaptateur.
- Ultra SCSI. (16 bits à 20 Mhz)
- Vitesse du bus d'E/S : 132 Mo/s.

Unités de stockage :

- 2 disques Ultra SCSI de 9, 1 Go par noeud (débit = 40 Mo/s).
- Externe : une batterie SSA (Sequentiel Storage Access) de 4 disques de 9, 1 Go.

Système d'exploitation :

- AIX4.3.3
- PSSP 3.1.1 (Parallel System Support Programs) pour AIX.

Compilateurs :

- XLC, XLC++, Fortran, Java.

Bibliothèques :

- de calcul : ESSL, P- ESSL, POSL.
- de communication : MPI, PVM, LAPI.
- de threads : pthreads, OpenMP.

Unité de contrôle :

- Station IBM 43P-140 munie d'un processeur PowerPc 604 à 233 Mhz.

II.3. Le système RS/6000 SP :

L'IBM RS/6000 SP est une famille de solutions de calcul parallèle. Elle est gérée par le système d'exploitation AIX (Unix IBM) avec le système de support de programmes parallèles (Parallel System Support Programs : PSSP) et ce sur l'unité de contrôle ainsi que sur les noeuds. L'architecture extensible du système SP est basée sur sa communication de haute performance et des processeurs PowerPC qui offrent un traitement aisé des données de grande

taille, des calculs intensifs et des entrée/sorties intensives. On peut exécuter simultanément des applications parallèles et séquentielles en dirigeant le système d'un seul poste de travail.

Plusieurs utilisateurs peuvent exécuter des requêtes complexes en manipulant de grands flots de données et obtenir leurs résultats de manière interactive.

L'architecture de réseau LANopen, basée sur le système d'exploitation AIX, permet l'intégration du système SP dans l'environnement existant. L'architecture du logiciel est adaptée à la configuration du système SP pour un usage maximal et performant. [redbooks].

II.4. Architecture :

II.4.1. Nœud :

Une RS/6000 SP est un assemblage de plusieurs noeuds. Un noeud contient des microprocesseurs, de la mémoire, des unités d'entrée/sortie, et un microprocesseur de service. Le tout est relié par un bus à haut débit. Chaque noeud est une machine parallèle à mémoire partagée, contenant quatre processeurs, une mémoire, une unité de stockage et des interfaces d'entrée/sortie ; elle est gérée par le système d'exploitation AIX.

II.4.2. Processeur :

Tous les processeurs sont des PowerPc 604e qui sont une implémentation 32-bits du PowerPc à architecture RISC (Reduced Instruction Set Code).

La technologie RISC est basée sur le fait que chacune des instructions du processeur est de longueur courte et fixe, et peut, la plupart du temps, se dérouler sur un cycle d'horloge, elle comporte 70 instructions.

Les opérations arithmétiques ne s'effectuent que sur les registres. Les informations sont seulement déplacées d'un registre à l'autre ceci afin d'en accélérer l'exécution et le calcul. Par ailleurs, les processeurs RISC peuvent faire plusieurs opérations en même temps et cela quelque soit la nature des instructions.

Un processeur RISC est moins volumineux, plus rapide et dégage moins de chaleur qu'un processeur CISC (Complex Instruction Set Code), tel un Pentium qui comporte 160 instructions, dans lequel l'exécution d'une instruction nécessite plus qu'un cycle d'horloge (architecture complexe).[jussieu]

Le PowerPc 604e est un processeur SMP (Symmetric MultiProcessor) à architecture super scalaire effectuant quatre instructions en un cycle d'horloge sur sept unités d'exécution indépendantes :

- 2 unités de calcul flottant.
- 3 unités de calcul entier.
- 2 unités de chargement/ déchargement.

II.4.3. Réseau :

La SP2 comporte un réseau standard à câble coaxial (débit 10 Mo) pour le contrôle et l'accès à la machine pour ne pas surcharger son réseau rapide qui est utilisé pour la communication inter-noeuds lors des exécutions des programmes parallèles. Le réseau rapide sert aussi à la connexion des noeuds d'extension.

Le SP-Switch est un réseau rapide qui a été conçu spécialement pour cette machine.Il offre seize connexions directes entre les seize noeuds de la machine avec un débit de 1Go. Dans le cas de notre SP on utilise un switcher rapide de 1 Mo.

II.5. Logiciels :

II.5.1. AIX :

AIX est le système d'exploitation UNIX adopté par IBM pour les grandes applications. Les fonctionnalités d'AIX parfaitement adaptées aux exigences industrielles ont fait leurs preuves au fil des ans dans une large gamme d'environnements de serveurs,allant de systèmes monoprocesseurs relativement petits aux serveurs évolutifs massivement parallèles RS/6000 SP.

AIX est reconnu comme le meilleur système d'exploitation Unix du marché pour les applications de gestion d'installations importantes et complexes. Il constitue une réelle valeur ajoutée en terme de fiabilité, de disponibilité et de sécurité. Il apporte un certain nombre d'améliorations relatives à Java, aux performances sur le Web et à l'évolutivité. De plus, il est conçu pour accueillir des applications Linux. AIX contient des outils d'administration à distance via le Web pour contrôler les ressources stratégiques, la disponibilité du système, des

cartes et du réseau. En fin, AIX assure la communication parfaite des applications 32 et 64 bits.

II.5.2. POE (Parallel Operating Environment) :

Le but de l'environnement parallèle (POE) est de permettre le développement et l'exécution des applications parallèles sur de multiples processeurs, appelés noeuds. POE définit un noeud maître qui dirige des interactions avec les utilisateurs et assure une gestion transparente de l'allocation des noeuds éloignés.

II.5.3. LoadLeveler :

IBM LoadLeveler pour AIX est un système de gestion de processus qui permet aux utilisateurs d'exécuter plus de processus en un temps plus court, tout en équilibrant les besoins des processus qui s'exécutent avec les ressources disponibles. LoadLeveler organise les processus et fournit des fonctions pour construire, soumettre, et développer les tâches rapidement et efficacement dans un environnement dynamique.

LoadLeveler permet aux utilisateurs de grouper les tâches d'un processus parallèle pour s'exécuter ensemble sur le même noeud, avec éventuellement un nombre différent de tâches sur les différents noeuds.

II.5.4. GPFS :

IBM GPFS (General Parallel File System) fournit les services d'un système de gestion fichiers pour les applications parallèles et séries qui s'exécutent sur la RS/6000 SP. GPFS offre aux utilisateurs des accès partagés aux fichiers qui peuvent couvrir plusieurs unités de disques sur de multiples noeuds SP. Le système GPFS, comparé à d'autres types de systèmes sur la RS/6000 SP, a les propriétés suivantes :

- Améliore la performance du système.
- Assure la consistance des fichiers.
- Augmente la disponibilité des données.
- Accroît la flexibilité du système.
- Simplifie l'administration.

L'une des plus importantes améliorations est celle qui se situe au niveau de l'environnement Message Passing Interface (MPI), qui définit un ensemble d'interfaces d'entrée/sortie pour l'usage des programmes parallèles partageant des données. GPFS fournit une série d'interfaces étendues qui permettent à la couche MPI de spécifier les caractéristiques d'accès à ses données et permet à GPFS d'optimiser son accès aux données.

II.5.5. IBM Virtual Shared Disk (Partage Virtuel de disque) :

IBM Virtual Shared Disk (VSD) est un logiciel qui permet aux noeuds de la RS/6000 SP de partager des disques avec les autres noeuds dans la même partition du système.Un partage virtuel de disque est un volume logique qui peut être accédé, non pas seulement par le noeud auquel il appartient, mais aussi par tout autre noeud supportant le logiciel VSD dans la partition du système.

Un serveur VSD est un noeud qui possède un nombre de VSD's. Il lit et écrit les données sur les VSD's selon les requêtes des noeuds clients. Un noeud client VSD est un noeud qui demande l'accès à un VSD. Un noeud peut être serveur et client en même temps.

II.5.6. Concurrent Virtual Shared Disks :

Concurrent Virtual Shared Disks inclut un accès disque concurrent qui permet l'utilisation de plusieurs serveurs pour satisfaire les requêtes de disque en profitant de l'avantage de l'environnement de l'accès concurrent de disque fourni par AIX. Pour utiliser cet environnement, VSD utilise les services du gestionnaire de volume logique concurrent (Concurrent Logical Volume Manager : CLVM).

Les avantages du VSD peuvent se résumer comme suit :

- Meilleure distribution des ressources système.
- Allocation dynamique de la mémoire.

III. Environnements parallèles

La machine dispose d'un certain nombre d'environnements parallèles automatique et d'optimisation des performances de communication.

III.1. MPI (Message Passing Interface) :

MPI est une librairie C portable permettant d'échanger des messages entre processus répartis sur un ensemble de processeurs.

III.2. OpenMP :

L'interface logicielle OpenMP est utilisée avec les architectures à mémoire partagée. OpenMP est un standard définissant un ensemble de directives et de fonctions permettant de gérer le partage du calcul entre plusieurs processeurs.

III.3. PVM :

PVM est constitué d'une bibliothèque de routines pouvant être appelées dans des programmes C ou Fortran. Elles permettent, entre autre, à l'utilisateur de définir une machine à mémoire distribuée à partir d'un ensemble de machines scalaires, parallèles ou vectorielles.

III.4. LAPI (Low Level Application Interface) :

Le LAPI constitue un protocole de communication (IBM), permettant d'optimiser les performances de la machine en terme de communication.

IV. Implémentation de l'algorithme DD :

IV.1. Processus de communication :

Comme nous l'avons déjà vu dans le second chapitre l'algorithme DD adopte une communication collective de type ALL_TO_ALL, c'est-à-dire que chaque processeur transmet ses données au autres y compris à lui-même et reçoit des données de tout les processeurs y compris ses propre données. Voici ci-dessous une illustration de ce type de communication (voir figure 5).

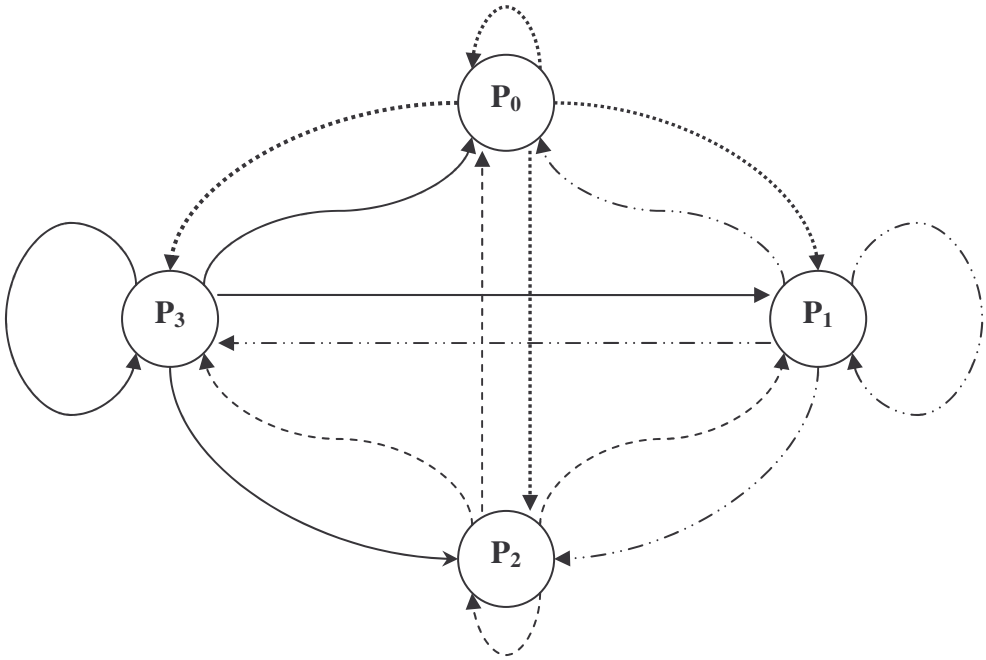


Figure 5: ALL_TO_ALL communication

IV.2. Structures de données:

- *ITEMSET* : une liste chaînée contenant deux champs de données .Un premier (*item*) de type chaîne de caractère et un deuxième (*app*) de type entier.

- *ITEMFREQ* : une liste chaînée contenant un seul champ (*item*) de données de type chaîne de caractère.

- *ITEM* : un enregistrement composé de deux champs, un premier (*item*) de type chaîne de caractère et un deuxième (*app*) de type entier.

- *ITEMF* : un enregistrement composé d'un champ unique (*item*) de type chaîne caractère.

Les champs *item* et *app* contiennent respectivement le code et le nombre d'apparition de l'item.

- *L* : Une liste de *ITEMSET* qui sert à stocker les items fréquents de taille 1.

- *C* : Une liste de *ITEMSET* qui sert à stocker les items candidats à chaque itération de l'algorithme.
- *LF* : Une liste de *ITEMFREQ* qui sert à stocker les items fréquents de taille k générés par chaque processeur à la k^{mme} itération.

- *LFC* : Une liste de *ITEMFREQ* qui sert à stocker l'ensemble des items fréquents global construit à partir des ensembles locaux.

- *Itemtab* : tableau à deux dimensions d'*ITEM* de *nb_procs* lignes et *taille_tab* colonnes.

- *Itemtabfreq* : tableau à deux dimensions d'*ITEMF* de *nb_procs* lignes et *taille_tab* colonnes.

Ces différents tableaux sont utilisés pour l'échange de données inter-processeurs puisque l'architecture de notre machine SP2 ne permet pas la transmission de liste chaînée ainsi nous sommes, à chaque fois, obligés de copier le contenu de la liste chaînée , que nous voulons transmettre , dans un tableau.

- *Tab_buf* : tableau à deux dimensions de buffers de *nb_procs* lignes et *taille*+1 colonnes.

- *Tab_cand* : tableau deux dimensions de chaîne de caractères contenant les noms des fichiers candidats respectifs à chaque processeur.

- *Tab_nom* : tableau deux dimensions de chaîne de caractère contenant les noms de fichiers de bases de données respectif à chaque processeurs.

Chaque processeur accède au fichier candidat et au fichier base de données relatif à son identifiant.
Avec *nb_procs* : nombre de processeurs, *taille_tab* : nombre d'item différents et taille : taille du buffer.

- **Les commandes de création des types MPI :**

  - ***MPI_Type_contigous( )*** : une fonction qui permet de créer une structure de données à partir d'un ensemble homogène de type prédéfini de données contiguës en mémoire [IDRIS].

    ***Syntaxe*:**

    *MPI_Type_contiguous (count, oldtype, *newtype)* [MHPCC].

  - ***MPI_Type_struct ( )***: le sous programme *MPI_Type_struct ( )* est le constructeur de type le plus général. Il permet la réplication de blocs de données de types différents [IDRIS].

    ***Syntaxe*:**

    *MPI_Type_struct(count,blocklens[],offsets[],old_types,*new_type)* [MHPCC].

  - ***MPI_Type_commit ( )*** : A chaque fois que l'on crée un type de données, il faut le valider à l'aide de la fonction *MPI_type_commit ( )* [IDRIS].

    ***Syntaxe*:**

    MPI_Type_commit (datatype) [MHPCC].

- **Les commandes MPI :**

  - ***MPI*_Allgather** : cette commande permet de faire en une seule opération une série de communication point à point entre les processus du communicateur (*MPI_COMM_WORLD*).

***Syntaxe*:**

*MPI_Allgather (*sendbuf, sendcount, sendtype, *recvbuf, recvcount, recvtype, MPI_COMM_WORLD)* [MHPCC].

sendbuf : buffer d'emission.

sendcount : nombre de blocks émis.

sendtype : type de données des blocks émis.

recvbuf : buffer de réception.

recvcount : nombre de blocks reçus.

recvtype : type de données des blocks reçus.

MPI_COMM_WORLD : communicateur par défaut.

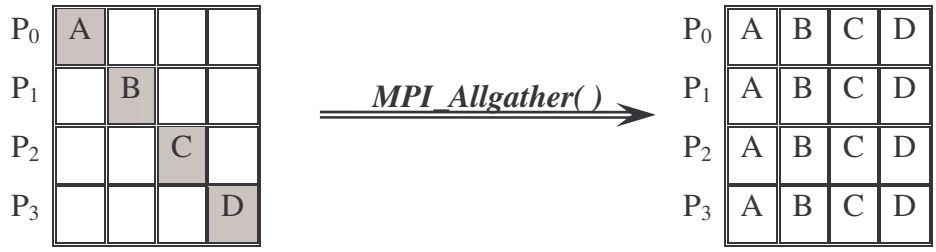


Figure 6 : Collecte générale *(MPI_Allgather)*

- ***MPI_Barrier*** : cette fonction assure les synchronisations globales entre les processus du communicateur (*MPI_COMM_WORLD*).

***Syntaxe*:**

MPI_Barrier (MPI_COMM_WORLD) [MHPCC].

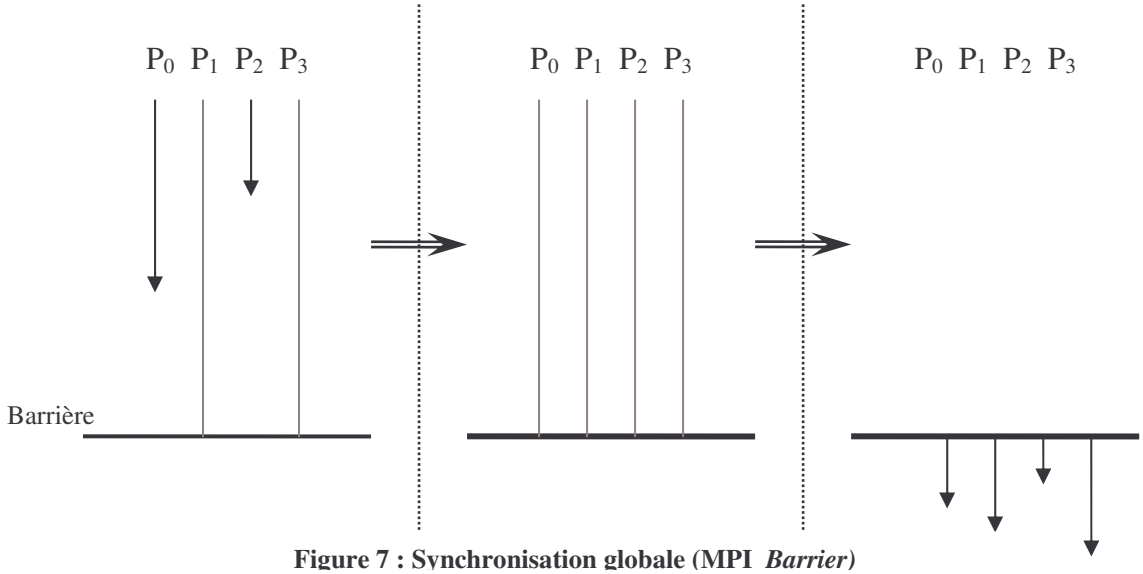


Figure 7 : Synchronisation globale (MPI *Barrier)*

IV.3. Fonction principale de l'algorithme DD :

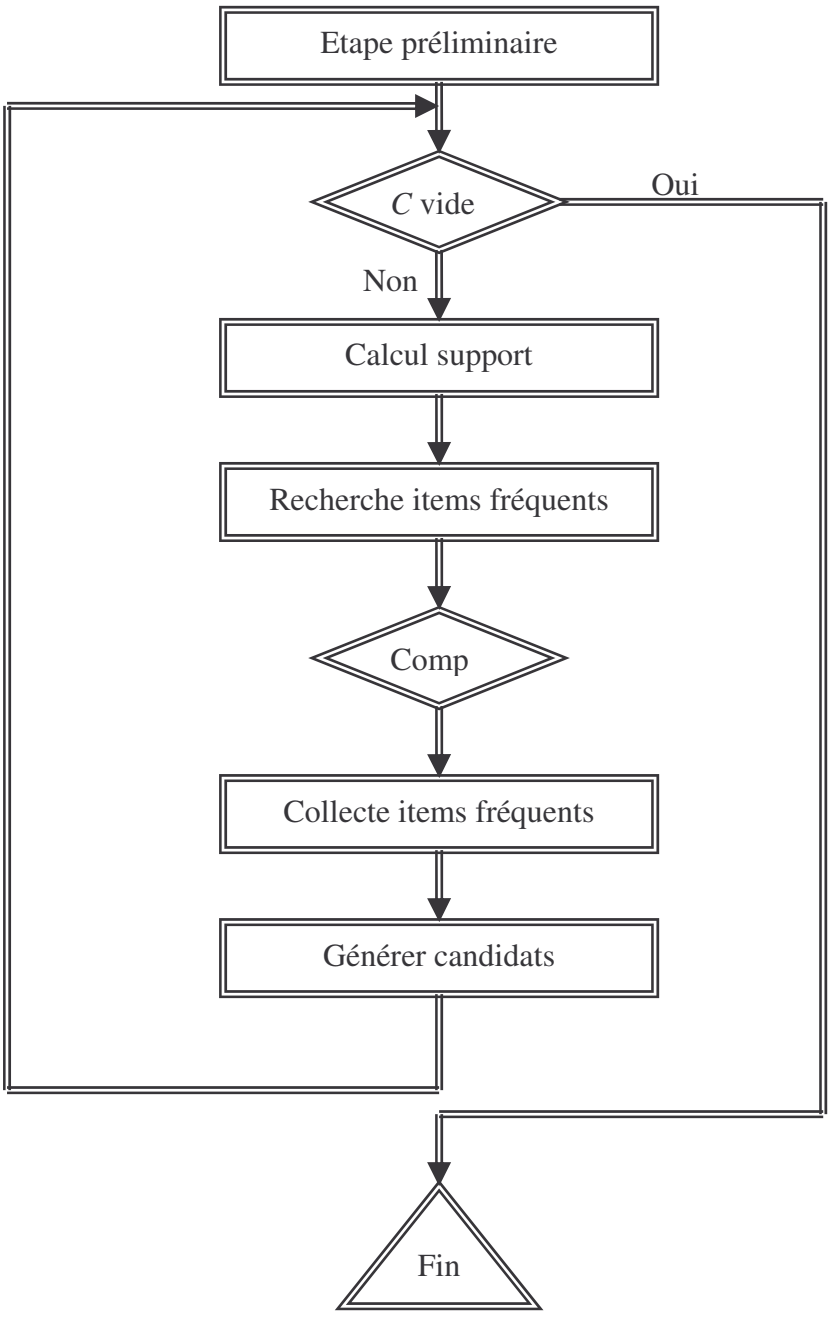


Figure 8 : Algorithme DD

Le test **C vide** est vraie si la liste *C* ne contient aucun élément, faux sinon.

IV.3.1. Fonction Etape préliminaire :

Cette étape est spéciale elle est exécutée une seule fois afin de générer l'ensemble des items candidats de taille 2 qui représente le point de départ de l'algorithme. Voici ci-dessous le déroulement de cette fonction (voir figure 9).

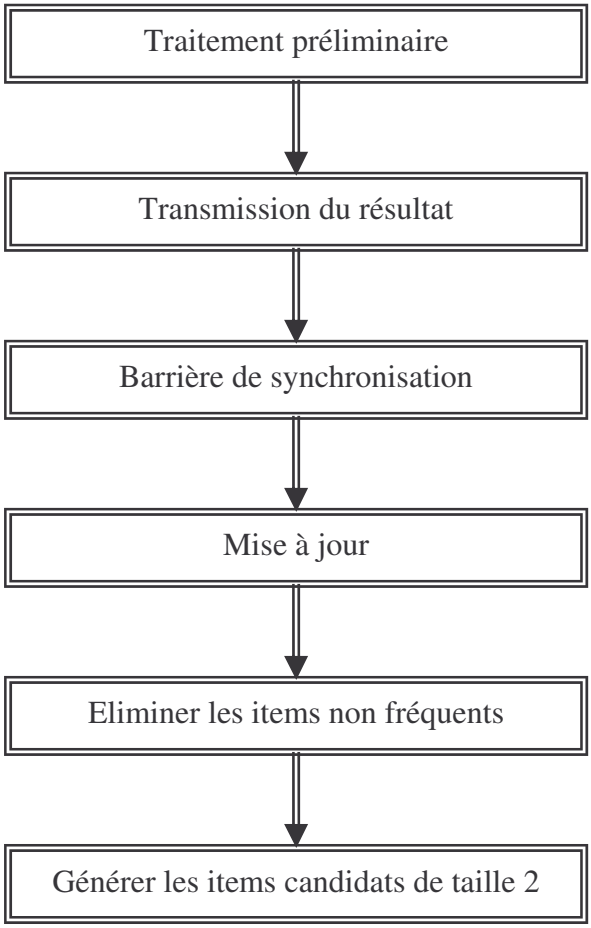


Figure 9 : Etape préliminaire (DD)

- **Traitement préliminaire :**

Cette fonction sert à extraire de la base de données l'ensemble des items candidats de taille 1 et de les stocker dans la liste chaînée **L** selon un ordre croissant. Voici ci-dessous le déroulement de cette fonction (voir figure 10)

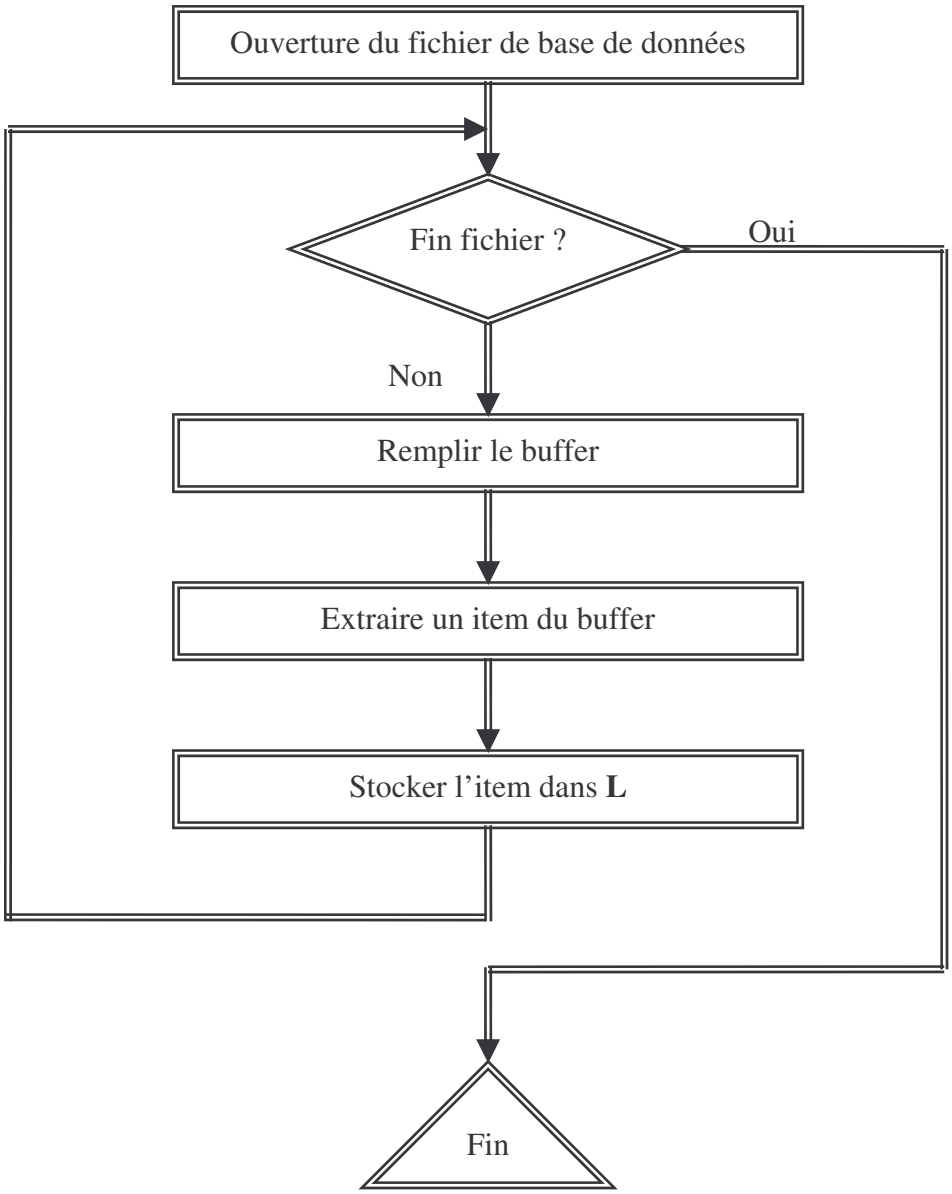


Figure 10 : Traitement préliminaire DD

- **Transmission du résultat :** Au cours de la première fonction de l'étape préliminaire, chaque processeur a déjà déterminé l'ensemble des *1-itemsets_candidats* avec bien évidemment leur support locaux respectifs. Dans cette fonction chaque processeur va maintenant transmettre son calcul locale à tout les autres. Pour cela nous devons tout d'abord copier la liste **L** dans le tableau *Itemtab* relatif à l'identifiant du processeur correspondant et par la suite transmettre ce tableau au autres processeurs par le billet de la primitive *MPI_Allgather ( ).*

- **Barrière de synchronisation :** Chaque processeur doit attendre la fin de toutes les opérations de transmission. Dans le cas où cette barrière ne serait pas respectée nous risquons d'avoir des résultats erronés.

- **Mise à jour :** L'opération de mise à jour consiste à accéder au tableau *Itemtab* reçu à partir de chaque processeur et, pour chaque item, effectuer l'opération suivante : si l'item existe déjà dans la liste **L** faire une mise à jour du support sinon stocker l'item avec son support dans la position adéquate.

- **Eliminer les items non fréquents :** Cette opération consiste à parcourir la liste **L** ainsi formée après les trois premières opérations (de calcul, de transmission et de mise à jour) et éliminer les items dont le support est inférieure à *minsup*.

- **Générer les items candidats de taille 2 :** Cette opération est triviale comparée à la fonction de génération des candidats de tailles quelconques utilisée dans les itérations suivantes. En effet la génération des *2-itemsets_candidat* se fait en une seule étape (joins tep), nous n'avons pas besoin d'exécuter la seconde étape (prune step) car nous sommes sûr que chaque sous ensemble de chaque item est fréquent.

IV.3.2. Fonction Calcul Support :

C'est dans cette fonction que se manifeste l'apport du parallélisme, elle comporte la phase de communication entre les processeurs avec toutes les synchronisations nécessaires pour assurer la cohérence des résultas et la phase de traitement effectué par chaque processeur qui consiste à accéder à la base de données et à calculer le support des items candidats afin de déterminer l'ensemble des items fréquents (voir figure 11).

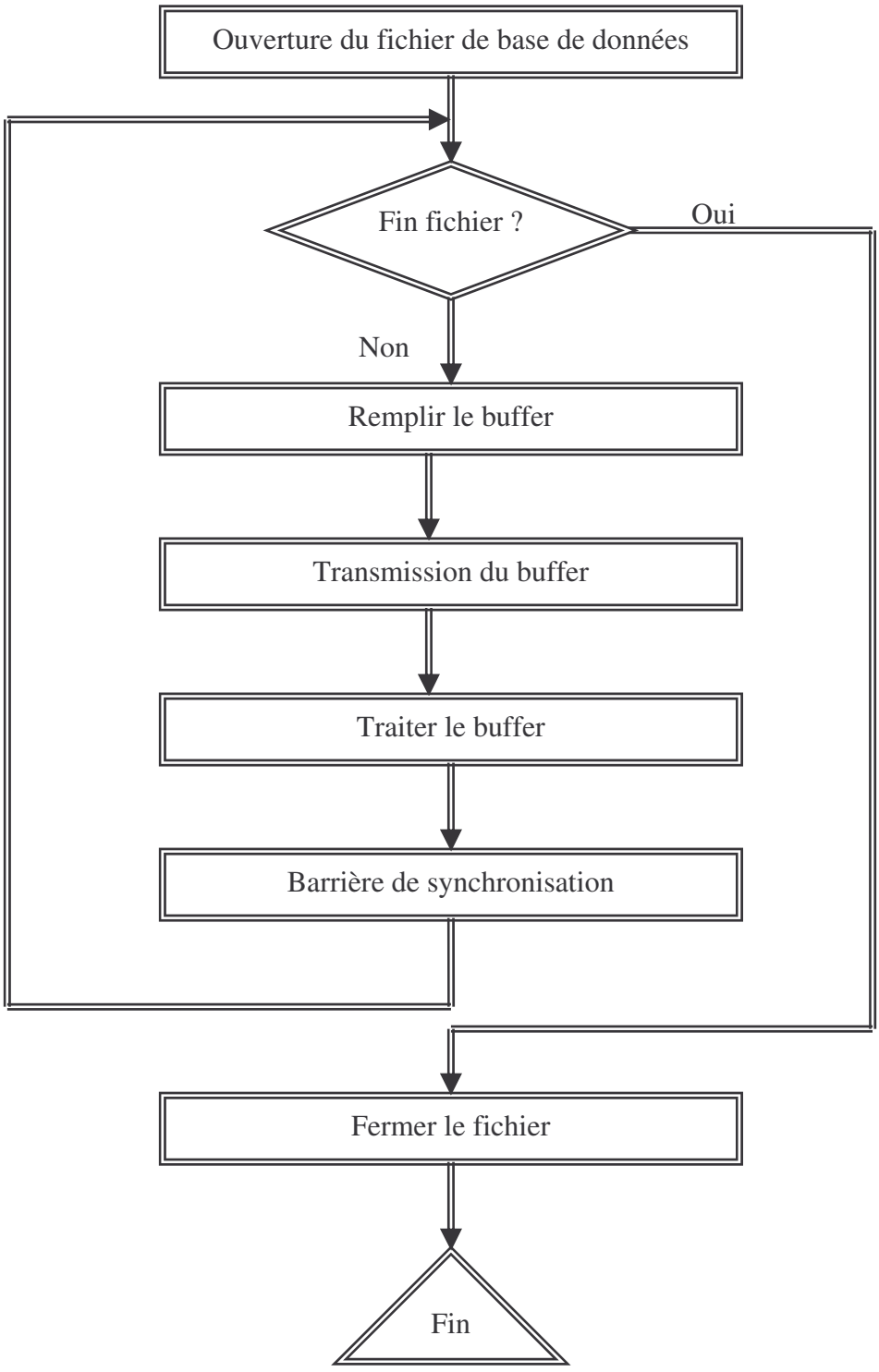


Figure 11 : Calcul support (DD)

- **Ouverture du fichier de base de données :** chaque processeur ouvre, en mode lecture, le fichier de base de données, relatif à son identifiant.

- **Fin fichier ? :** le traitement de la fonction **calcul support** se termine quand ce test est vraie.

- **Remplir le buffer :** À chaque itération, chaque processeur remplis un buffer à partir de sa portion de base de données stockée dans sa mémoire.

- **Transmission du buffer :** Chaque processeur transmet le buffer à tous les autres en utilisant (*MPI_Algather ( )*).

- **Traiter le buffer :** chaque processeur compte le nombre d'apparition de chacun des items candidats à partir de l'ensemble de buffers.

- **Barrière de synchronisation :** cette barrière garantie la synchronisation entre les processeur, en effet chaque processeur ne recommence de nouveau (à partir de la fonction **remplir buffer**) que si tous les autres ont terminés leurs traitement. Cela est nécessaire puisque si un des processeurs termine avant les autres il remplit son buffer puis il le transmet, cette opération écrase le contenu des buffers ce qui risque de fausser les résultas.

IV.3.3. Fonction Recherche des items fréquents :

Dans cette fonction nous faisons un parcours de la liste *C*, un test est effectué sur chaque élément de la liste : si le nombre d'apparitions est supérieur au support minimum, l'item est stocké dans la liste *LF* sinon nous passons à l'élément suivant.

Le test **Comp** est vrai lorsque le nombre de candidats (*nbr_cand*) est supérieur au nombre de processeurs (*nb_procs*).

IV.3.4. Fonction Collecte des items fréquents :

Dans cette fonction chaque processeur va maintenant transmettre son calcul locale à tout les autres. Pour cela nous devons en premier lieu copier la liste **LF** dans le tableau *Itemtabfreq* relatif à l'identifiant du processeur correspondant, en deuxième lieu transmettre ce tableau au autres processeurs via la primitive *MPI_Allgather ( )* et enfin en troisième lieu stocker cet ensemble ainsi formé des items fréquents globaux dans la liste chaînée **LFC**. Voici ci-dessous une illustration en diagramme de cette fonction (voir figure 12).

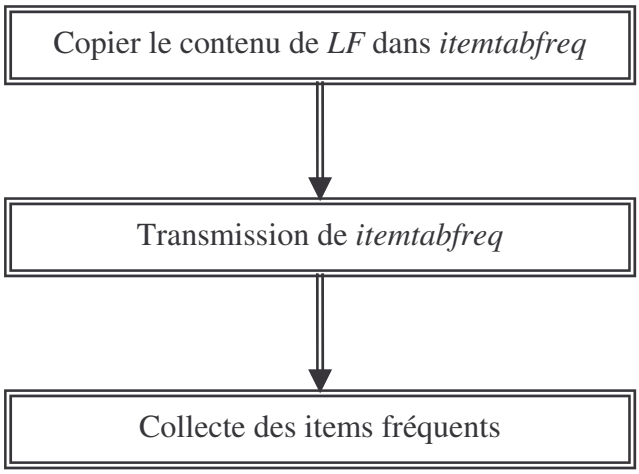


Figure 12 : Collecte des items fréquent (DD)

IV.3.5. Fonction *Générer candidats* :

A la k^{emme} itération cette fonction génère à partir de LFC les items candidats de taille k+1 dans la liste C, cela se fait comme nous l'avons déjà expliqué dans le deuxième chapitre. Ensuite chaque processeur divise la liste C ainsi construite pour ne garder que la portion relative à son identifiant cette portion sera forcément unique pour chaque processeur.

V. Implémentation de l'algorithme IDD :

V.1. Processus de communication :

Grâce à cette nouvelle technique de communication en anneau, nous avons remplacé les communications collectives où tous les processeurs transmettent leurs données locales aux autres processeurs par deux communications point à point entre voisins. Voici ci-dessous une illustration de ce processus de communication (voir figure 13).

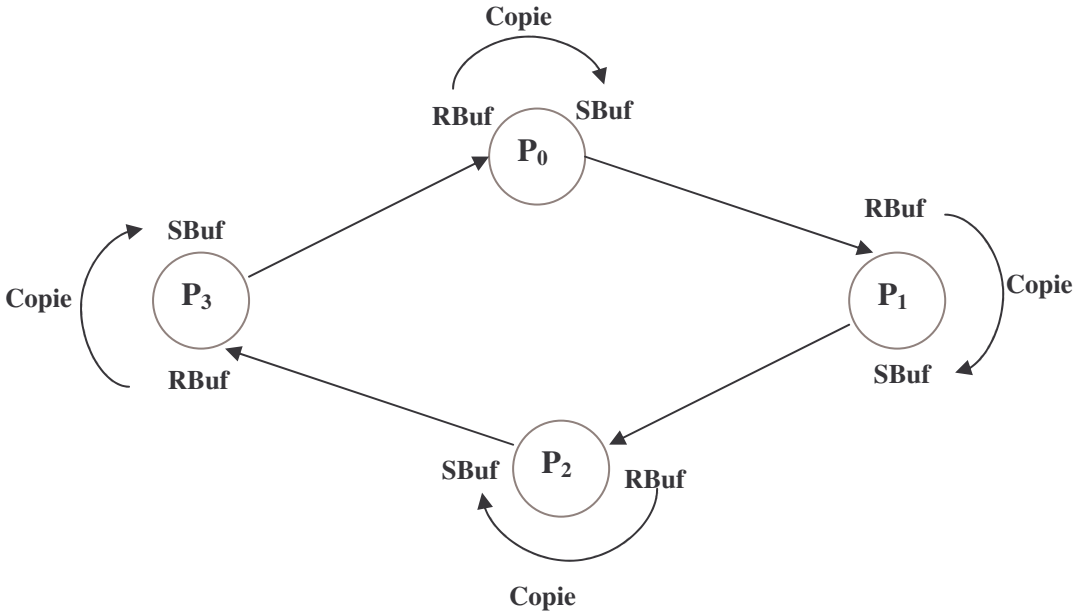


Figure 13 : Communication en anneau logique

V.1. Structures de données :

Comme nous l'avons déjà illustré, la différence entre cet algorithme et l'algorithme DD réside dans la phase de communication ainsi nous n'avons changé que les structures de données utilisées pour assurer l'échange des données. Par conséquent dans IDD nous n'avons plus besoin de tableau *Tab_buf* qui sera remplacé par un buffer d'émission *SBuf* et un buffer de réception *RBuf.*

Suite à cette modification dans les structures des données et le déroulement du processus de communication, nous n'avons plus besoin d'utiliser *MPI_Algather*. Nous avons donc opter pour l'utilisation des directives de communications asynchrones non bloquantes *MPI_Isend* et *MPI_Irecv*.

***Syntaxe* :**

- *MPI_Isend (*buffer, count, datatype, destination, tag, comm,*request)* [MHPCC].

buffer : buffer d'émission.

count : nombre.

datatype : type des données émises.

destination : processeur destination.

tag : drapeau du message.

comm : MPI_COMM_WORLD.

request : variable de retour qui indique l'état de la transmission.

- *MPI_Irecv (*buffer, count, datatype, source, tag, comm, *request)* [MHPCC].

buffer : buffer d'émission.

count : nombre.

datatype : type des données émises.

source : processeur émetteur.

tag : drapeau du message.

comm : MPI_COMM_WORLD.

request : variable de retour qui indique l'état de la transmission.

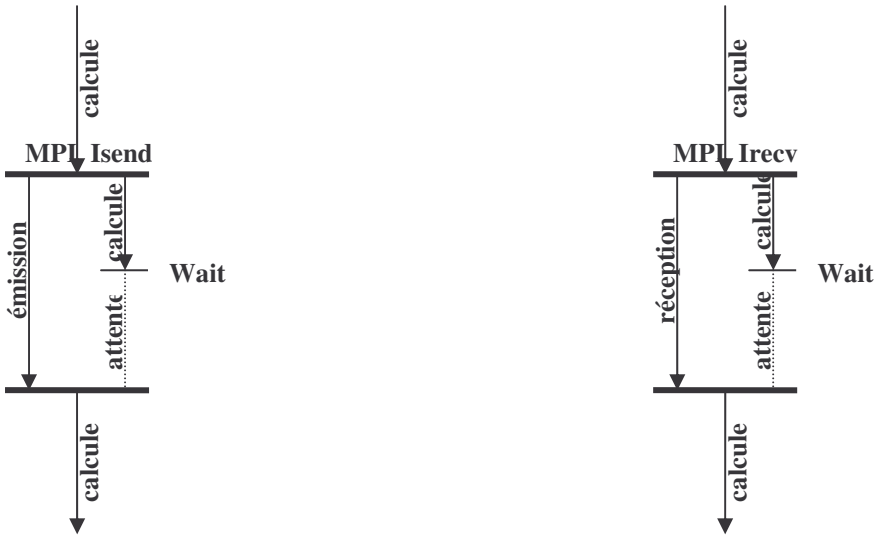


Figure 14 : Envoi non bloquant avec recopie temporaire du message (MPI_*Isend)*

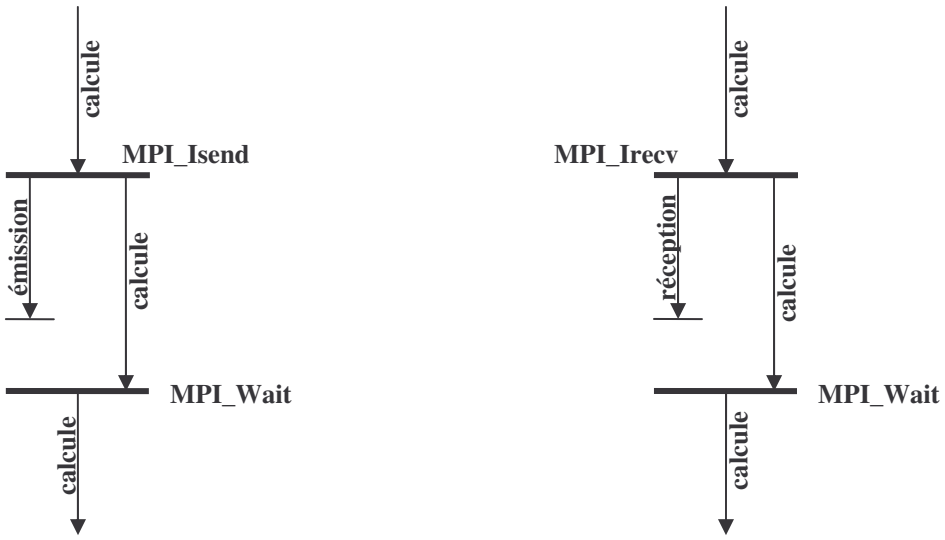


Figure 15 : Envoi non bloquant avec recopie temporaire du message (MPI_*Isend)*

Le chevauchement calcul/transmission constitue la plus importante caractéristique et l'un des meilleurs avantages fournis par l'utilisation des primitive *MPI_Isend* et *MPI_Irecv*. Les deux figures ci-dessus illustrent deux cas de chevauchement calcul/transmission de données. Dans le premier cas (voir figure 14), la phase de transmission de données prend plus de temps que la phase de calcul les processeurs sont donc obligés d'attendre la fin de la communication pour reprendre le calcul ils sont alors bloqués en exécutant la commande *MPI_Waite*. Dans le deuxième cas (voir figure 14), en exécutant la commande *MPI_Wait* les processeurs trouvent que la communication est déjà terminée, il n'y a donc pas de temps d'attente.

V.2. Fonction principale de l'algorithme IDD:

Après une étude détaillée sur l'algorithme IDD et son processus de communication nous avons constater qu'il suffit de modifier la fonction **Calcul Support** qui assure l'échange de données tout en gardant la même structure de l'algorithme DD.

V.2.1. Fonction Calcul Support :

Comme pour l'algorithme DD cette fonction est la plus importante et la plus coûteuse, en temps de calcul, nous verrons dans le chapitre suivant que ce nouveau mode de communication nous a permis d'améliorer considérablement les accélérations et les temps de

réponse obtenus par DD. Voici ci-dessous un diagramme qui illustre cette fonction (voir figure 16).

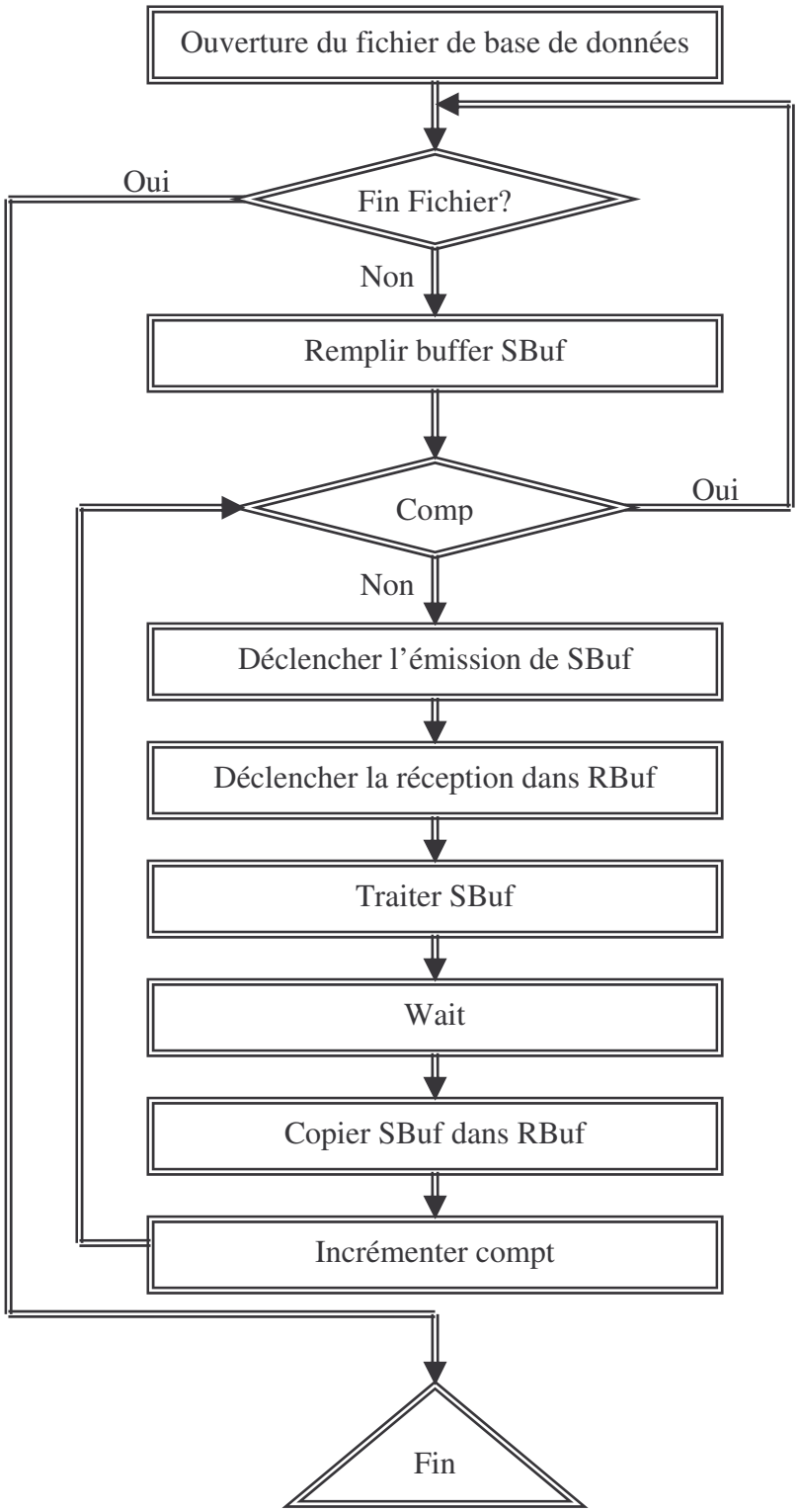


Figure 16 : Calcul support (IDD)

- **Ouverture du fichier de base de données :** chaque processeur ouvre le fichier de base de données correspondant à son identifiant.

- **Remplir buffer SBuf :** identique à celle de DD.

- **Déclencher l'émission de SBuf :** déclencher une opération d'émission asynchrone non bloquante par le billet de la directive *MPI_Isend*.

- **Déclencher la réception dans RBuf :** déclencher une opération de réception synchrone non bloquante par le billet de la directive *MPI_Irecv*.

Ces deux directives *MPI_Isend* et *MPI_Irecv* permettent de chevaucher de l'échange de données et du calcul en effet le processeur déclanche la communication et continue l'exécution du code sans attendre la terminaison de celle-ci. Il ne se bloque que s'il atteint la primitive *MPI_Wait*, qui elle est bloquante, avant que la communication ne soit terminée.

- **Traiter SBuf :** identique à celle de DD.

- **Wait :** le processeur doit attendre la terminaison des opérations de communication avant de copier RBuf dans SBuf pour ne pas écraser le contenu de ce dernier, ce qui risquerait de fausser les résultats.

- **Copier SBuf dans RBuf :** les processeurs sont vus comme étant un anneau logique c'est pourquoi nous devons copier RBuf dans SBuf pour le transmettre au processeur suivant.

- **Incrémenter compt :** compter P-1 opération de communication.

VI. Conclusion :

Nous venons, dans ce chapitre d'exposer notre implémentation des deux algorithmes DD et IDD avec les choix que nous avons fait en ce qui concerne les structures de données et les directives MPI qui ont été utilisées pour assurer la communication entre les processeurs.

Nous allons maintenant dans le chapitre suivant exposer les différents résultats obtenus par ces deux algorithmes en faisant varier et le nombre de processeurs et le support minimum (*minsup*), interpréter ces résultats, donnés par chaque algorithme à part, et bien évidemment comparer DD et IDD de point de vue temps de réponse et accélération.

Evaluations expérimentales

I. Introduction :

Après le chapitre précèdent où nous avons présenté les structures de données, les directives de parallélisme et la démarche utilisée pour l'implémentation des deux algorithmes DD et IDD, nous allons, en première partie de ce chapitre, illustrer et interpréter les différentes mesures obtenues pour chacun des deux algorithmes. Une comparaison entre les deux familles DD et IDD selon différents métriques fait l'objet d'une deuxième partie et enfin, une synthèse qui récapitule les performances des deux algorithmes.

II. Exécution séquentielle :

Cette exécution est obtenue en déroulant un des deux programmes DD ou IDD sur un seul processeur de la machine SP2 puisque la différence entre ces deux programmes réside seulement dans la communication.

Les mesures obtenues par cette exécution sont nécessaires pour pouvoir calculer les paramètres de parallélisme tel que l'accélération et l'efficacité afin d'interpréter les différents résultats obtenus dans ce qui suit.

II.1 Mesures et Interprétation :

En diminuant le support, l'algorithme DD génère plus d'items fréquents ce qui se répercute par une augmentation du temps d'exécution. En effet, il est évident que pour tout algorithme séquentiel le temps d'exécution augmente avec nombre d'opérations effectuées (voir tableau 7 et figure 17).

Support	10%	9%	8%	7%	6%	5%
temps exécution	206,75	233,13	355,69	758,11	1960,99	4495,86

Tableau 7 : Temps d'exécution séquentiel

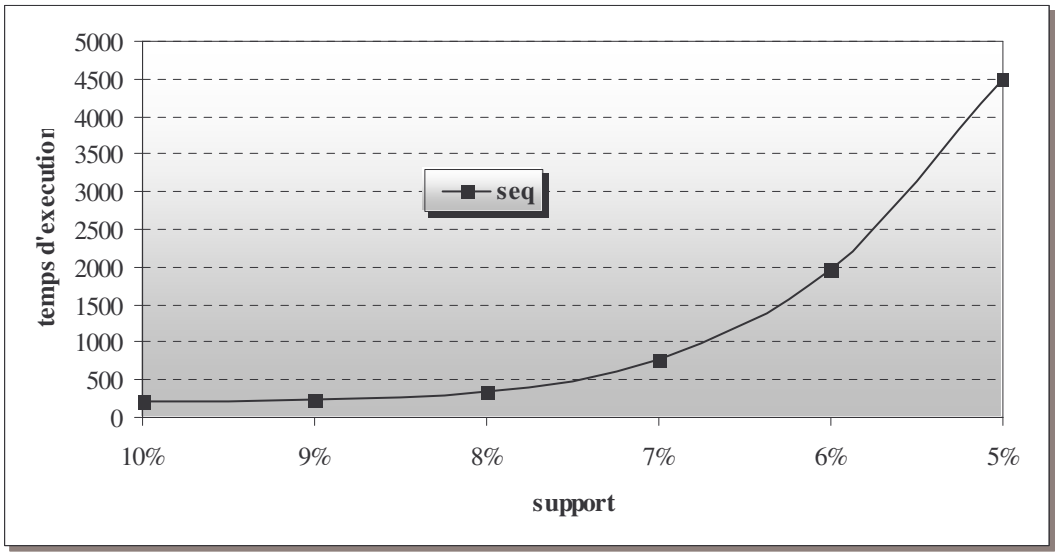


Figure 17 : Courbe d'exécution sur un seul processeur.

III. Exécution parallèle de l'algorithme DD :

III.1. Mesures :

Le tableau 8 résume les mesures obtenues pour l'exécution parallèle de l'algorithme DD en variant le nombre de processeurs et le support.

support	10%	9%	8%	7%	6%	5%
Temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Deux processeurs	119,49	137,46	220,71	484,08	1265,32	2989,49
Quatre processeurs	62,34	76,76	125,45	274,71	737,81	1767,27
Huit processeurs	130,43	139,83	162,43	241,21	498,63	1073,55
Douze processeurs	243,41	245,95	280,51	307,31	472,09	866,58
Seize processeurs	356,44	360,11	379,79	400,75	560,44	855,11

Tableau 8 : Temps d'exécution parallèle de DD.

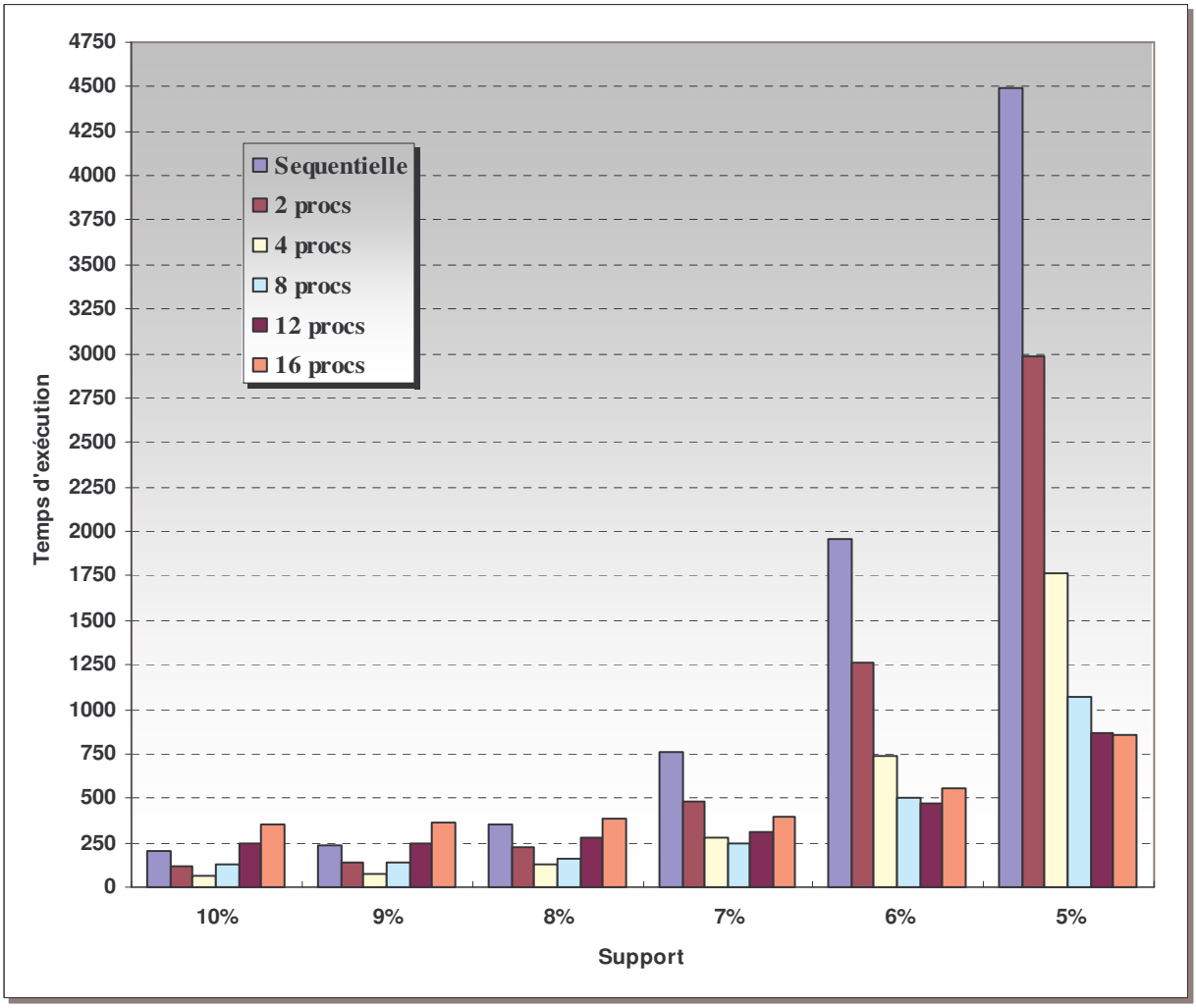


Figure 18 : Histogramme d'exécution parallèle de DD.

III.2. Interprétation :

Pour des supports relativement élevés (10%,9% et 8%), la part de la communication est largement plus importante que celle du traitement. En effet d'après l'histogramme ci-dessus (voir figure 18), nous remarquons que le meilleur temps d'exécution pour ces supports est obtenue pour quatre processeurs. Cela est dû au faite que les quatre processeurs choisis appartiennent au même nœud ; par conséquent la communication entre eux se fait à travers la mémoire partagée alors que pour huit, douze et seize processeurs l'échange de données se fait à travers le réseau ce qui augmente nettement le temps de réponse. D'autant plus que, le mode de communication adopté par l'algorithme DD (*all-to-all brodcast*) pénalise considérablement les temps de transmissions de données.

Par contre, pour 7% la part de traitement augmente mais elle reste relativement faible pour mettre en évidence l'apport du parallélisme. En effet, d'après l'histogramme ci-dessus (voir figure 18), les meilleurs temps de réponse sont donnés par l'exécution sur huit processeurs. De même, pour 6% bien que la part de traitement augmente considérablement, il est encore inutile d'augmenter le nombre de processeurs jusqu'à seize, en effet l'histogramme ci-dessus (voir figure 18) nous permet de voir que l'exécution sur douze processeurs donne le meilleur résultat.

En fin, en diminuant le support jusqu'à 5% l'apport de parallélisme est nettement perceptible. En effet les mesures illustrées par l'histogramme ci-dessus (voir figure 18) montrent bien qu'il est intéressant d'augmenter le nombre de processeurs jusqu'à seize.

IV. Exécution parallèle de l'algorithme IDD :

IV.1. Mesures :

support	10%	9%	8%	7%	6%	5%
temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Deux processeurs	120,04	138,37	222,58	483,95	1264,64	2986,23
Quatre processeurs	62,47	76,26	126,14	276,78	736,27	1762,72
Huit processeurs	59,61	70,24	100,80	160,38	424,13	996,74
Douze processeurs	91,53	92,46	127,44	139,67	310,20	717,04
Seize processeurs	138,74	145,49	144,56	159,47	310,54	600,89

Tableau 9: Temps d'exécution parallèle de IDD.

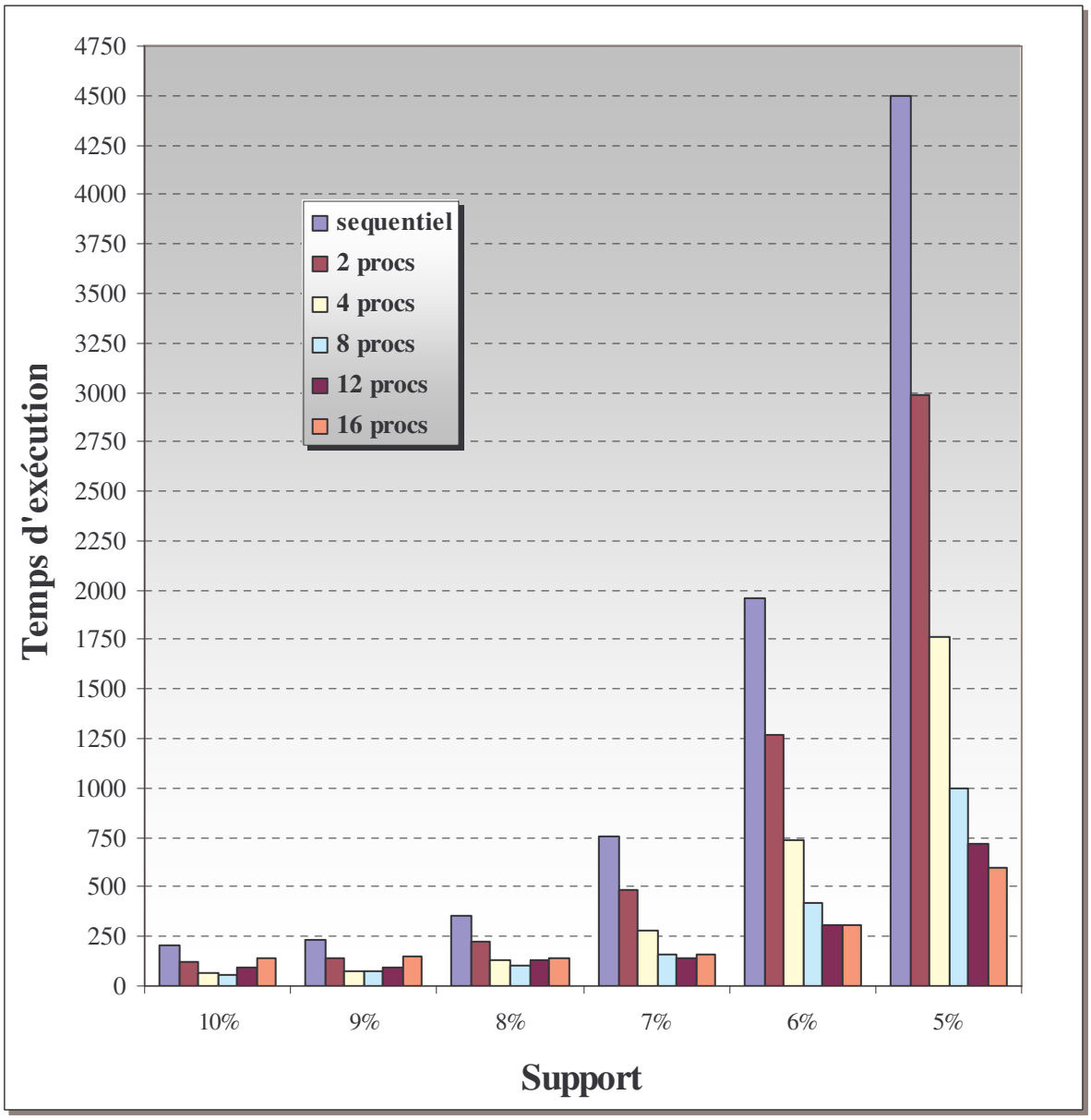


Figure 19 : Histogramme d'exécution parallèle de IDD

IV.2. Interprétation :

Comme nous l'avons déjà vu pour l'algorithme DD, la part de traitement est nettement plus faible que celle de la communication pour les supports relativement élevés (10%, 9% et 8 %). Quoi que pour l'algorithme IDD le mode de communication adopté (en anneau logique) est meilleur que celui de DD (voir paragraphe suivant). Ainsi, l'histogramme ci-dessus (voir figure 19) montre que les meilleurs temps de réponse sont donnés par l'exécution sur huit processeurs mais pas sur quatre comme le cas de DD.

En diminuant le support d'un cran jusqu'à 7%, il devient intéressant d'augmenter le nombre de processeurs jusqu'à douze mais pas plus car la part de traitement n'est pas encore assez importante.

Par contre, pour 6% et 5% il devient avantageux d'allez jusqu'à seize processeurs. En effet, les mesures illustrées dans l'histogramme ci-dessus (voir figure 19), montre que l'exécution sur seize processeurs donne le meilleur temps de réponse.

V. Comparaison entre DD et IDD :

V.1. Cas de deux processeurs :

support	10%	9%	8%	7%	6%	5%
Temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Temps parallèle DD	119,49	137,46	220,71	484,08	1265,32	2989,49
Temps parallèle IDD	120,04	138,37	222,58	483,95	1264,63	2986,22

Tableau 10 : DD et IDD sur 2 processeurs.

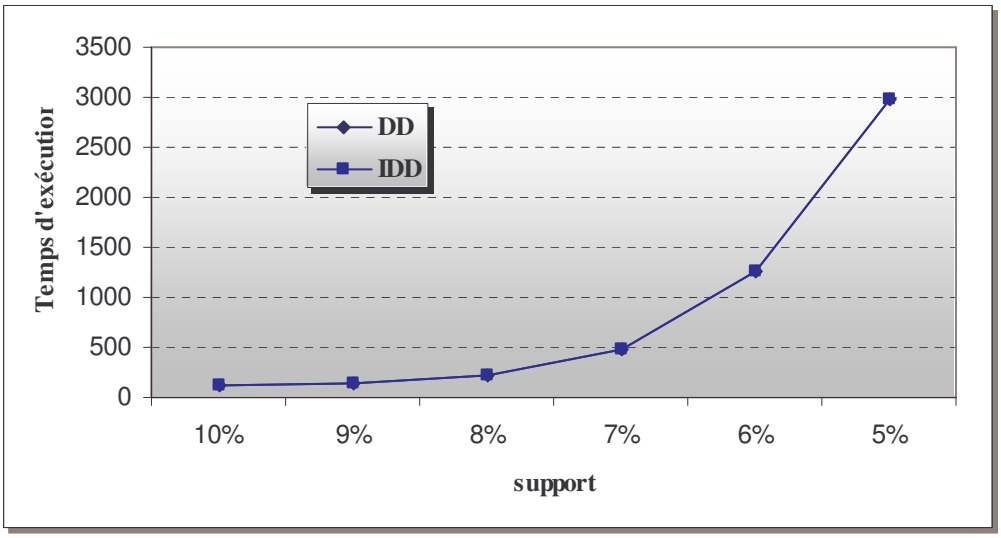


Figure 20 : Courbes d'exécution de DD et IDD sur 2 processeurs.

Pour ces mesures sur deux processeurs, nous en avons choisi deux du même nœud. Puisque la différence entre DD et IDD réside seulement dans la communication, nous avons obtenus, à quelques millisecondes près, les même résultats (voir figure 20).

support	10%	9%	8%	7%	6%	5%
Accélération DD	1,73	1,70	1,61	1,57	1,55	1,50
Efficacité DD	0,87	0,85	0,81	0,78	0,77	0,75
Accélération IDD	1,72	1,68	1,60	1,57	1,55	1,51
Efficacité IDD	0,86	0,84	0,80	0,78	0,78	0,75

Tableau 11 : Caractéristiques de DD et IDD sur 2 processeurs.

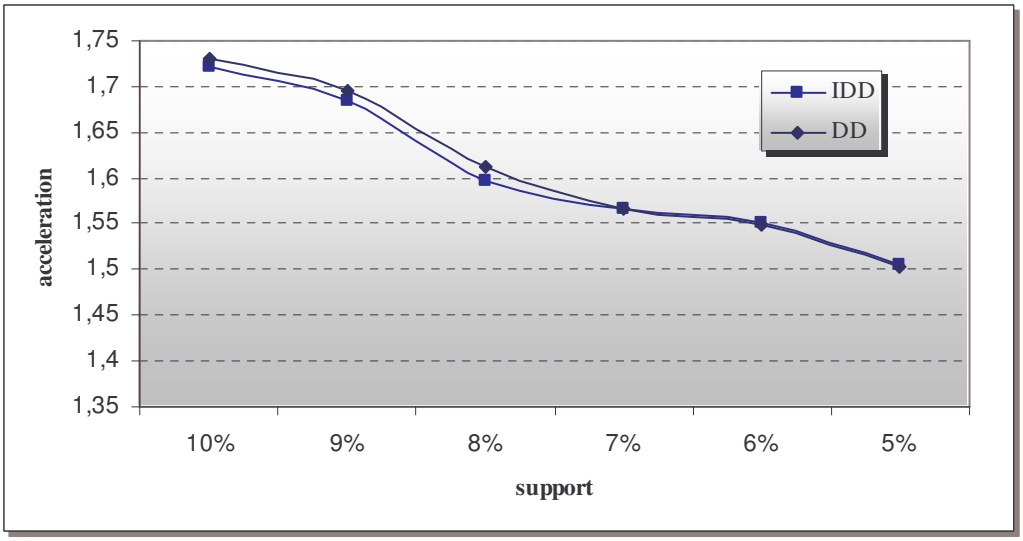


Figure 21 : Courbes d'accélération de DD et IDD sur 2 processeurs.

Puisque les temps d'exécution de DD et IDD sont à peu prés les mêmes, nous avons obtenu deux courbes d'accélération presque confondues (voir figure 21).

V.2 Cas de quatre processeurs :

support	10%	9%	8%	7%	6%	5%
Temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Temps parallèle DD	62,34	76,76	125,45	274,71	737,81	1767,27
Temps parallèle IDD	62,47	76,26	126,14	276,78	736,27	1762,72

Tableau 12 : DD et IDD sur 4 processeurs.

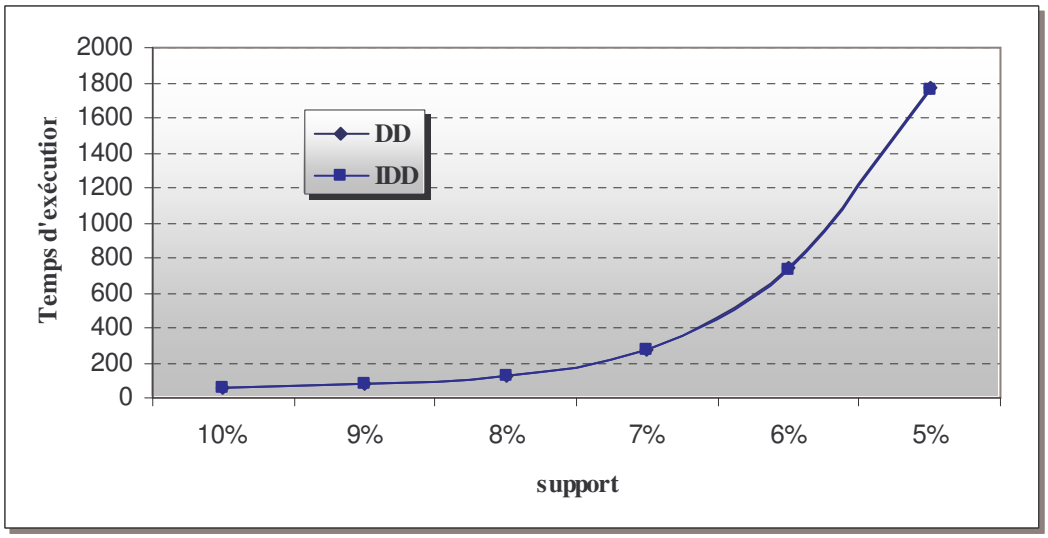


Figure 22 : Courbes d'exécution de DD et IDD sur 4 processeurs

support	10%	9%	8%	7%	6%	5%
Accélération DD	3,32	3,04	2,84	2,76	2,66	2,54
Efficacité DD	0,83	0,76	0,71	0,69	0,66	0,64
Accélération IDD	3,31	3,06	2,82	2,74	2,66	2,55
Efficacité IDD	0,83	0,76	0,70	0,68	0,67	0,64

Tableau 13 : Caractéristiques de DD et IDD sur 4 processeurs.

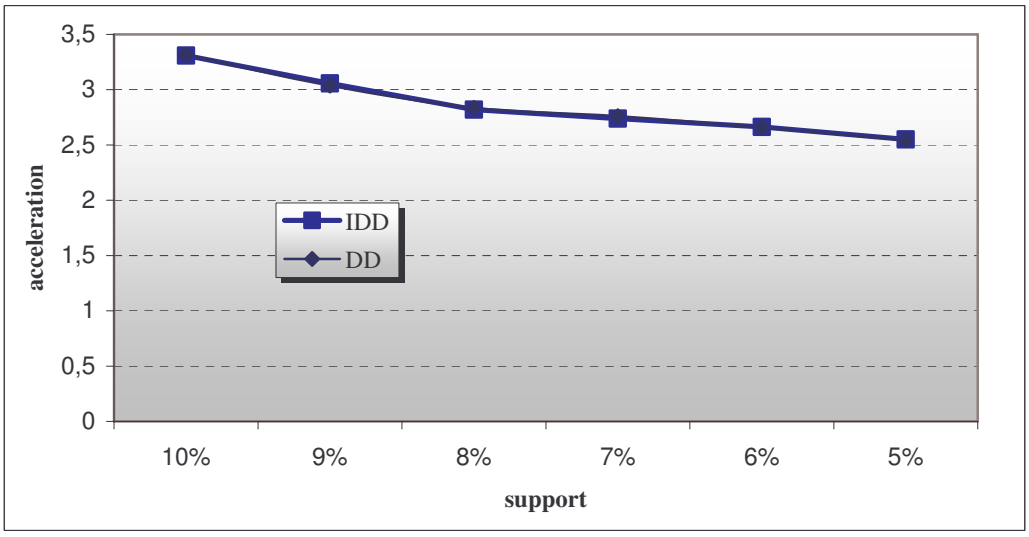


Figure 23 : Courbes d'accélération de DD et IDD sur 4 processeurs.

Pour les mêmes raisons présentées pour deux processeurs, les courbes d'exécution et d'accélération sont confondues (voir figure 22 et figure 23).

V.3. Cas de huit processeurs :

support	10%	9%	8%	7%	6%	5%
Temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Temps parallèle DD	130,43	139,83	162,43	241,21	498,63	1073,55
Temps parallèle IDD	59,61	70,24	100,80	160,38	424,13	996,74

Tableau 14 : DD et IDD sur 8 processeurs.

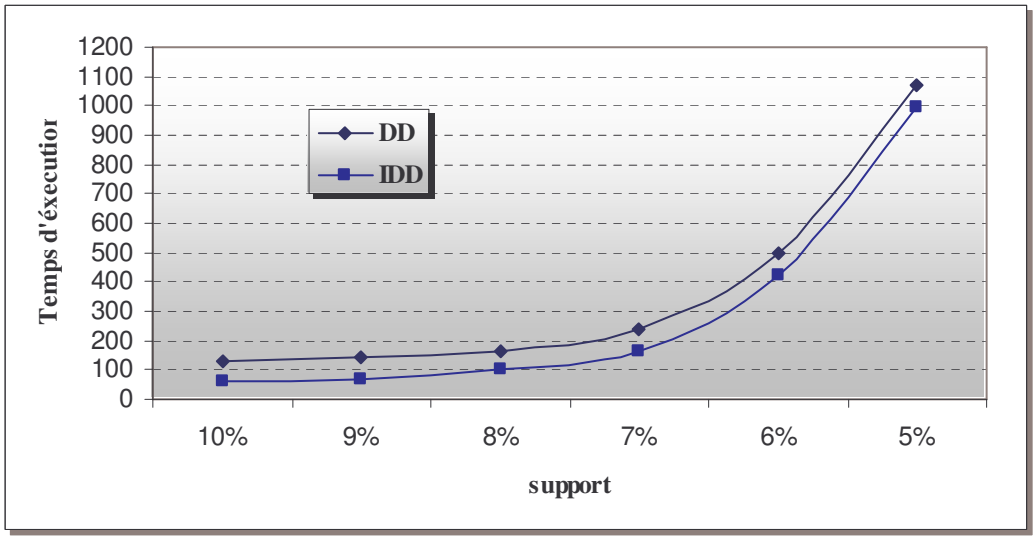


Figure 24 : Courbe d'exécution de DD et IDD sur 8 processeurs.

Les courbes d'exécution de DD et IDD sur huit processeurs (voir figure 24) montrent que, grâce à l'optimisation apportée à IDD, les temps d'exécution donnés par ce dernier sont inférieurs à ceux de l'algorithme DD. Cette différence n'est pas assez importante puisque sur huit processeurs, le mode de communication médiocre adopté par DD n'altère que légèrement les performances de cet algorithme.

support	10%	9%	8%	7%	6%	5%
Accélération DD	1,59	1,67	2,19	3,14	3,93	4,19
Efficacité DD	0,20	0,21	0,27	0,39	0,49	0,52
Accélération IDD	3,47	3,32	3,53	4,73	4,62	4,51
Efficacité IDD	0,43	0,41	0,44	0,59	0,58	0,56

Tableau 15 : Caractéristiques de DD et IDD sur 8 processeurs.

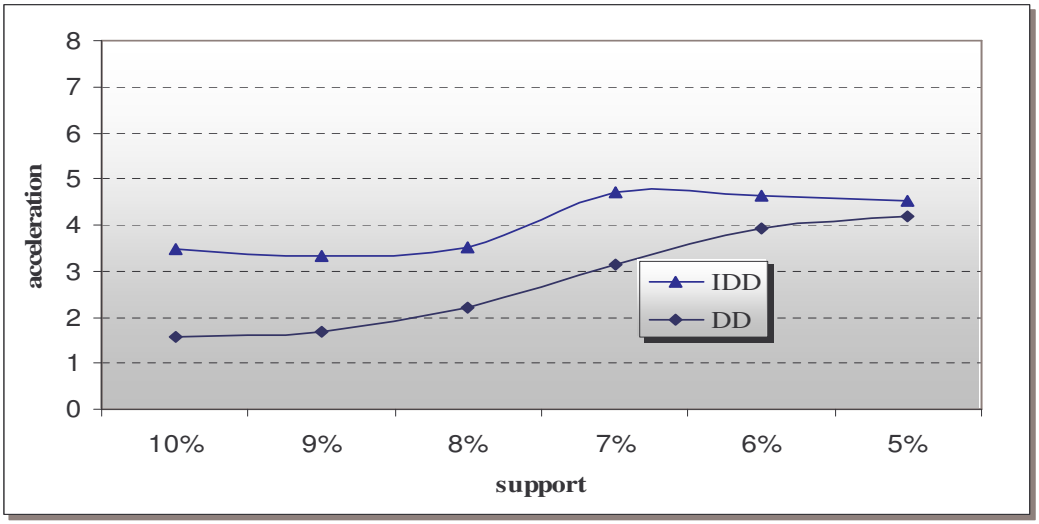


Figure 25 : Courbes d'accélération de DD et IDD sur 8 processeurs.

Comme nous l'avons vue précédemment, pour les supports faibles (10%, 9% et 8%), la part de la communication est plus importante que celle du traitement ; par conséquent l'accélération de l'algorithme DD est assez médiocre (1.59, 1.67, 2.19). Cette accélération s'améliore de plus en plus jusqu'à atteindre (4.19) à (5%).

Cependant pour l'algorithme IDD, grâce à un mode de communication assez performant, nous arrivons à atteindre des accélérations de (3.47) pour des supports relativement élevés tel que (10%). Cette accélération, comme pour le cas de DD vas bien sûr augmenter et dans ce cas elle atteint (4.51) pour (5%).

Les deux courbes d'accélération de DD et IDD (voir figure 25) illustrent clairement l'apport de IDD au niveau de la communication. En effet la différence entre les accélérations de DD et IDD est plus importante pour les supports élevés (10%) que pour des supports faibles (5%).

V.4. Cas de douze processeurs :

support	10%	9%	8%	7%	6%	5%
Temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Temps parallèle DD	243,41	245,95	280,51	307,31	472,09	866,58
Temps parallèle IDD	91,53	92,46	127,44	139,67	310,20	717,04

Tableau 16 : DD et IDD sur 12 procs

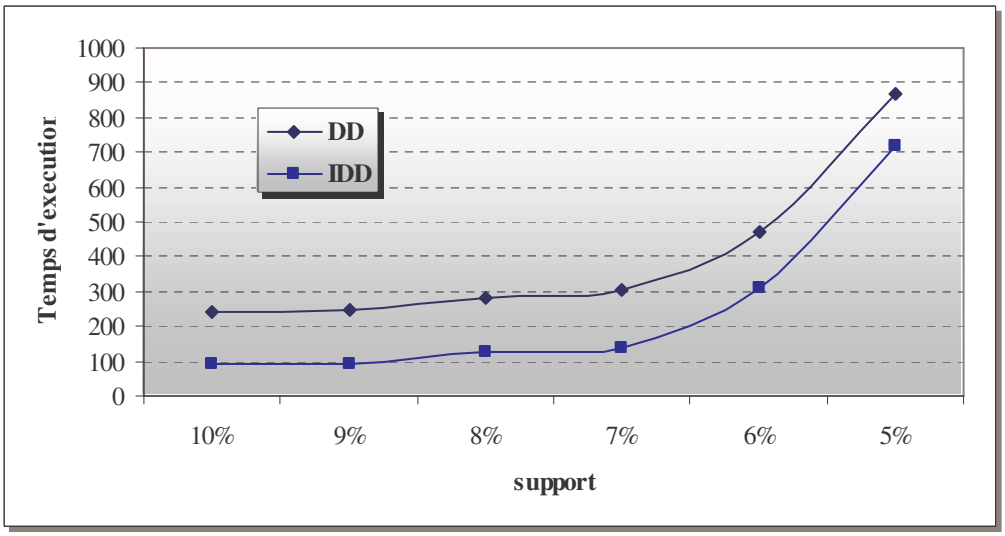


Figure 26 : Courbes d'exécution de DD et IDD sur 12 processeurs

L'exécution sur douze processeurs comporte plus de communication que celle sur huit. Par conséquent nous pouvons percevoir beaucoup mieux l'amélioration qu'apporte IDD sur le temps d'exécution.

support	10%	9%	8%	7%	6%	5%
Accélération DD	0,85	0,95	1,27	2,47	4,15	5,19
Efficacité DD	0,07	0,08	0,11	0,21	0,35	0,43
Accélération IDD	2,26	2,52	2,79	5,43	6,32	6,27
Efficacité IDD	0,19	0,21	0,23	0,45	0,53	0,52

Tableau 17 : Caractéristiques de DD et IDD sur 12 processeurs.

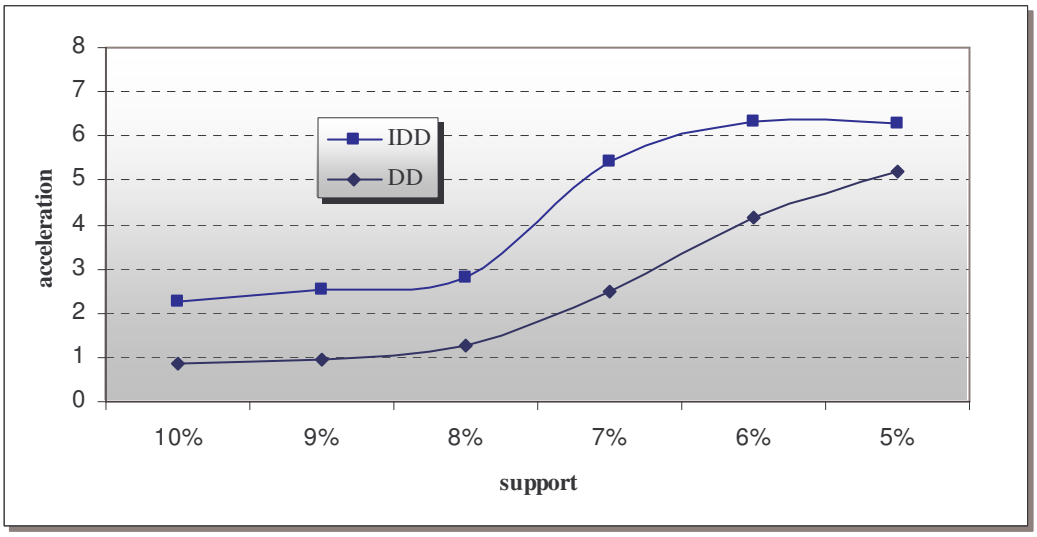


Figure 27 : Courbe d'accélération de DD et IDD sur 12 processeurs.

Les deux courbes ci-dessus d'accélération de DD et IDD sur douze processeurs (voir figure 27) montrent que, pour des supports relativement élevés, DD donne une très faible accélération : il est même moins performant que l'algorithme séquentiel (0.85), cela confirme bien ce que nous avons déjà illustrés. Concernant IDD, nous remarquons que, même avec des parts de communication élevées et des parts de traitement très faibles, l'accélération est de l'ordre de (2.26).

Il est clair que IDD donne des résultats meilleurs que ceux donnés par DD surtout lorsqu'il y'a plus de communication.

Pour les supports faibles (6%,5%), nous remarquons d'après les deux courbes d'accélération (voir figure 27) que la différence entre l'accélération donnée par DD et celle donnée par IDD diminue, cela est dû au fait que la part de traitement devient assez importante vis-à-vis la part de communication.

V.5. Cas de seize processeurs :

support	10%	9%	8%	7%	6%	5%
Temps séquentiel	206,75	233,13	355,69	758,11	1960,99	4495,86
Temps parallèle DD	356,44	360,11	379,79	400,75	560,44	855,11
Temps parallèle IDD	138,74	145,49	144,56	159,47	310,54	600,89

Tableau 18 : DD et IDD sur 16 processeurs.

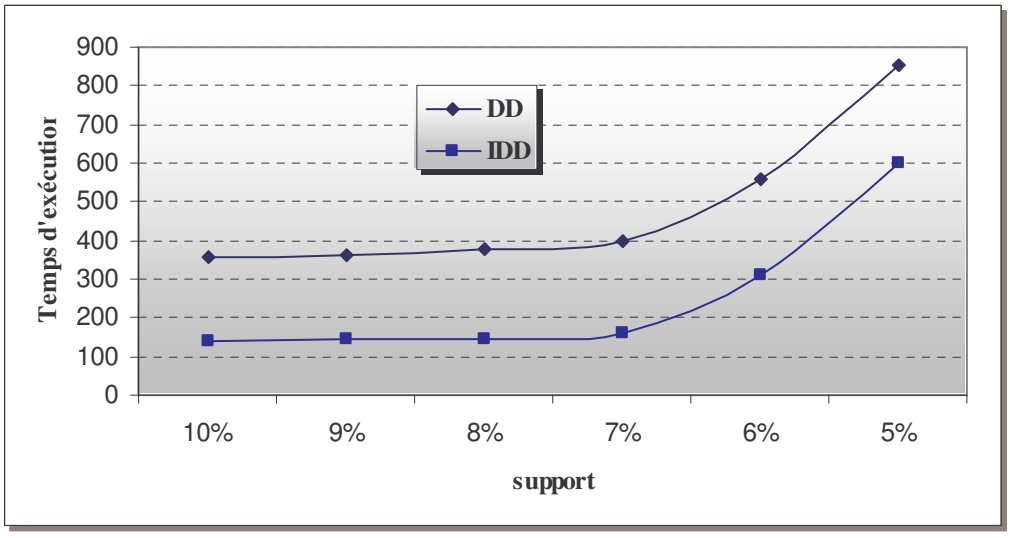


Figure 28 : Courbe d'exécution de DD et IDD sur 16 processeurs.

En nous referant au tableau 18, nous remarquons que les temps d'exécution donnés par l'algorithme IDD n'ont jamais dépassé les temps séquentiels, ce qui n'est pas le cas pour DD. En effet, pour le support 10% le temps d'exécution parallèle donné par DD est supérieur au temps séquentiel, cela semble être erroné, mais en examinant le problème de plus près nous nous rendons compte que cela était en fait prévisible car comme nous l'avons déjà mentioné pour les supports élevés la part de communication est beaucoup plus importante que celle du traitement. De plus DD adopte un mode d'échange de donnée peu performant c'est pourquoi il donne des résultats médiocres pour ces support qualifiés d'élevés. Nous remarquons aussi que cette défaillance de DD par apport à l'algorithme séquentiel a été déjà vue pour douze processeurs (206.75 – 243.41) mais elle n'était pas assez évidente que dans ce cas-ci (206.75 – 356.44); cela est, bien évidemment, dû au fait que l'augmentation du nombre de processeurs pénalise les temps de communication surtout dans les cas ou l'exécution est très pauvre en traitement. Ces anomalies n'existent pas pour IDD grâce, une autre fois, au mode d'échange de donnée adopté c'est-à-dire en anneaux logique.

De plus comme dans le cas de douze processeurs les différences entre les temps de réponse des deux algorithmes traités DD et IDD sont de plus en plus importantes (voir figure 28), chose que nous avons déjà prévue en illustrant le cas de l'exécution sur huit et douze processeurs (voir figure 28).

support	10%	9%	8%	7%	6%	5%
Accélération DD	0,58	0,65	0,94	1,89	3,50	5,26
Efficacité DD	0,04	0,04	0,06	0,12	0,22	0,33
Accélération IDD	1,49	1,60	2,46	4,75	6,31	7,48
Efficacité IDD	0,09	0,10	0,15	0,30	0,39	0,47

Tableau 19 : Caractéristiques de DD et IDD sur 16 processeurs.

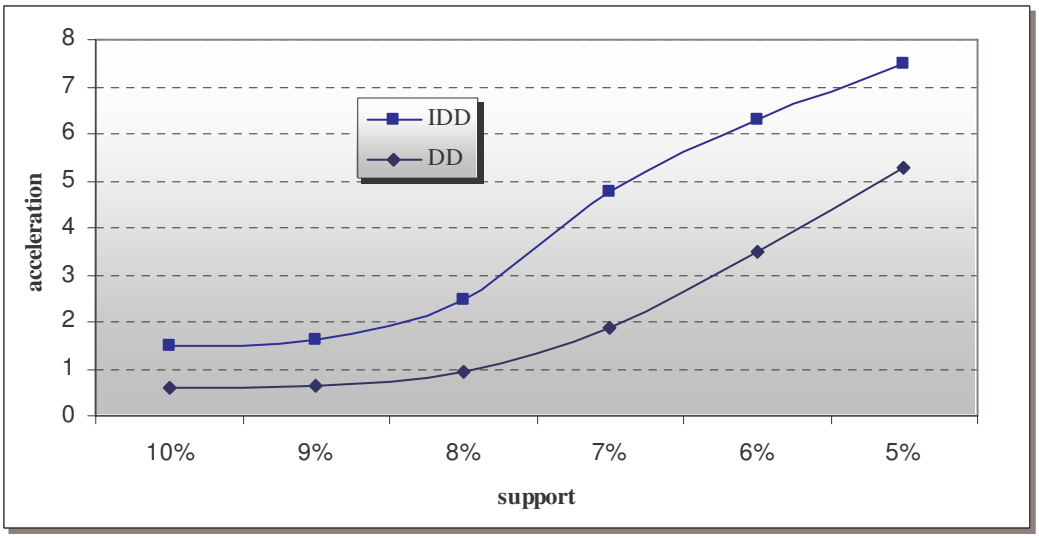


Figure 29 : Courbe d'accélération de DD et IDD sur 16 processeurs.

En nous referant au tableau 19, nous remarquons que les valeurs d'accélération et d'efficacité données par IDD sont nettement plus meilleures à celles données par DD. Cette amélioration s'explique par l'optimisation du processus de communication décrit par IDD.

Les deux courbes d'accélération présentées ci-dessus (voir figure 29) illustrent bien que les différences entre les performances des deux algorithmes seront plus perceptibles en augmentant le nombre de processeurs et en diminuant le support.

VI. Synthèse :

En ce qui concerne l'algorithme DD, nous remarquons que pour les supports dits élevés les meilleurs temps de réponse sont obtenus pour l'exécution sur quatre processeurs du même nœud. En effet, comme la communication est très importante par rapport au traitement et comme l'échange de données sur DD laisse beaucoup à désirer, il devient vraiment intéressant d'utiliser une des particularités de notre machine parallèle SP2 d'IBM (un nœud est vu comme étant une machine parallèle de quatre processeurs à mémoire partagée) afin de masquer cette défaillance de DD en assurant l'échange d'information à travers la mémoire partagée. Cela se limite au cas de supports élevés car pour des supports assez faibles (5%) la part de traitement augmente considérablement pour être exécutée sur quatre processeurs seulement. Il faut alors impérativement augmenter le nombre de processeurs sans trop se soucier de l'augmentation du taux d'échange de données car dans ce cas de support faible le traitement masque un peu la perte de performances dans la communication, qui reste malgré tout perceptible surtout en comparant DD à IDD.

De même, nous avons vu qu'en augmentant le nombre de processeurs sans pour autant diminuer le support, les performances de DD se dégradent à un point qu'il devient moins performant que l'algorithme séquentiel c'est à dire que si nous dépassons un certain seuil de communications inter-nœuds, l'algorithme séquentiel devient plus efficace que de l'algorithme DD. Cela représente encore une preuve du mauvais choix du mode de communication lors de la conception DD.

Pour IDD, nous remarquons que dés que nous passons de quatre à huit processeurs nous percevons clairement l'apport de cet algorithme qui se manifeste d'autant plus que le nombre de processeurs est élevé et que le support est faible c'est-à-dire que les taux de communications inter-nœud sont importants. En effet, pour deux ou quatre processeurs d'un même nœud, où la communication se fait à travers la mémoire partagée, l'accélération et l'efficacité décroisent avec le support. Alors que, pour huit processeurs l'accélération et l'efficacité varient selon le support ; cela est dû au fait qu'il y a un peu d'échange de données inter-nœud mais pas assez pour montrer les qualités de IDD. Par contre, à partir de douze processeurs, les communications inter-noeud deviennent plus importantes, ainsi nous pouvons observer, en diminuant le support, l'amélioration des performances (accélération et efficacité) de l'algorithme IDD.

Enfin, d'après les mesures et les remarques faites tout au long de ce chapitre, il est clair que IDD est bien plus performant que DD puisqu'il gère mieux les problèmes de communications inter-processeurs. Il nous a permis d'atteindre des accélérations que nous avons jugées acceptables.

VII. Conclusion :

Dans ce chapitre, nous avons mis l'accent sur la comparaison des deux familles DD et IDD en interprétant les différentes mesures obtenues à partir des différents tests effectués sur la machine parallèle SP2. Nous avons utilisé différentes métriques afin de mieux commenter notre comparaison.

Conclusion et Perspectives

Suite à une explosion vertigineuse de la quantité d'informations stockées dans les bases de données, il était nécessaire de recourir à des techniques et à des outils d'exploitation automatique de ces données, afin d'extraire l'information renfermée dans ces bases. Les techniques du datamining ont pour objectif de satisfaire cette nécessité, d'où l'énorme intérêt qu'elles ne cessent de susciter. Les algorithmes associés à ces techniques, dits algorithmes de recherche de règles d'associations, réalisent plusieurs passes sur la base de données qui sont synonymes de plusieurs accès disques, ce qui dégrade considérablement leurs performances. Ainsi, il y a eu plusieurs tentatives d'amélioration de ces techniques mais le gain en temps de réponse était insatisfaisant voir insignifiant. Il fallait

alors introduire de nouvelles approches, de ce fait le parallélisme semble être une voie prometteuse pour améliorer les performances de ces algorithmes, d'ou l'émergence de plusieurs familles d'algorithmes parallèles de découverte de règles associatives.

Ce projet a pour objectif principal d'étudier et d'implémenter deux algorithmes parallèles de Data Mining. Ainsi nous avons commencé ce travail par introduire le Data Mining, ses intérêts et ses différents domaines d'applications. Nous avons ensuite, dans une deuxième partie, présenté une analyse des principaux algorithmes séquentiels de découvertes des règles associatives suivie par une illustration de quelques versions parallèles de l'algorithme fondamental *Apriori*. La troisième partie était consacrée à l'implantation des deux algorithmes parallèles ***Data Distribution*** (**DD**) et ***Intelligent Data Distribution*** (**IDD**) accompagnée d'une présentation de l'environnement de travail. En dernière partie, nous avons effectué une étude expérimentale de ces deux algorithmes sur la machine parallèle SP2 d'IBM.

Apres avoir réalisé ces mesures expérimentales, et interprété les résultats donnés par **DD** et ceux donnés par **IDD**, il était claire que l'algorithme ***Intelligent Data Distribution*** était bien plus performant et plus efficace que l'algorithme ***Data Distribution***. Nous avons aussi constaté qu'avec les caractéristiques et l'architecture particulière de la machine SP2 nous ne pouvions pas atteindre de meilleures performances puisque, nous avons remarqué, que presque la moitié des ressources pour **IDD**, un peu plus pour **DD**, sont consommées par la communication et les accès disque. Ainsi avec d'autres machines parallèles plus performantes dotées de disques et de réseaux plus rapides, les deux algorithmes **DD** et **IDD** donneront des résultats plus intéressants.

Perspectives :

Parmi les perspectives qui nous semblent intéressantes, citons :

- Parallélisation des instructions de l'algorithme *Apriori*. En effet les approches parallèles existantes représentent presque toutes, une parallèlisation des données.
- Prévoir une nouvelle approche qui réduit le nombre d'accès au disque.

Bibliographie

[IBM1] : Planning, Volume 1, *Hardware and Physical Environment*.

[IBM2] : Planning, Volume 2, *Control Workstation and Software Environment*.

[RA & JS] : Rakesh Agrawal and Jhon Shafer. Parallel mining of association rules: Design, implementation and experience. Research Report RJ 10004, IBM Almaden Research Center, San Jose, California, February 1996.Available from http:// www.almaden.ibm.com/cs/quest.

[RA & RS] : R. Agrawal and R. Skirant. Fast algorithms for mining associations rules. In *Proceedings of the 20th Int*. conference on Very Large Databases, pages 478-499, June 1994.

[RA, TI & AS] : R. Agrawal, S. Gosch, T. Imielinski, B. Iyer, and A. Swami. An interval calssifer for databases mining applications. In *proceeding of the 18th Int*.

References Web:

[DMS] : Site http : //www.pmsi.fr/dminit2.htm.

[DME] : Site http : // www.math-appli-uco.fr/etudiants/projets/dataMining/entreprises.html.

[IDRIS] : Site http: // www.idris.fr/data/cours/parallel/mpi/mpi_cours.html.

[Jussieu] : Site http : //web.ccr.jussieu.fr/ccr/systeme.

[MHPCC] : Site http://www.mhpcc.edu/training/workshop/mpi/MAIN.html.

[redbooks] : Site http : //www.ibm.com/redbooks.

TABLE DES FIGURES

LISTE DES TABLEAUX