الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Systèmes Informatiques

Thème :

Conception d'un système basé sur le parallélisme pour l'extraction des radicaux

Encadré Par :

Dr.SOUSSI Hakim

Présenté par :

GUERGOUR Abdallah

Juillet 2019

Remerciements

La première personne que nous tenons à remercier est notre encadrant Dr. SOUSSI HAKIM pour l'orientation, la confiance, les conseils précieux qu'il nous a apporté tout au long de la réalisation de ce travail.

Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui nous ont enseigné et qui par leurs compétences nous ont soutenu

dans la poursuite de nos études.

Enfin, on remercie tous ceux qui, de près ou de loin, ont contribué

à la réalisation de ce travail.

Dédicace

Je dédie ce travail qui n'aura jamais pu voir le jour sans les soutiens indéfectibles et sans limite de mes chers parents qui ne cessent de me donner avec amour le nécessaire pour que je puisse arriver à ce que je suis aujourd'hui. Que dieux vous protège et que la réussite soit toujours à ma portée pour que je puisse vous combler de bonheur.

Je dédie aussi ce travail à :

-Mon frère, mes sœurs.

-Tous mes amis, mes collègues et tous ceux qui m'estiment.

RESUMÉ :

Ce mémoire s'inscrit dans le domaine de la recherche d'information (RI). Il a pour objet la conception d'un système pour l'extraction des radicaux basé sur le parallélisme en utilisant un algorithme séquentiel de radicalisation on le transformant en un algorithme parallèle on se basant sur les différentes techniques du parallélisme, afin d'améliorer les performances du processus d'indexation en termes de temps. Le mémoire est structuré en quatre chapitres : Le premier chapitre présente les concepts clés de la recherche d'information, notamment l'indexation des documents. Le deuxième chapitre est consacré au parallélisme. Le troisième et le quatrième chapitre présentent la conception du système ainsi que son implémentation.

MOTS-CLÉS: Parallélisme, radical, indexation, recherche d'information.

Sommaire

Liste des Figures

Liste des tableaux

Introduction Générale

La RI concerne les méthodes et mécanismes qui permettent la création et l'utilisation d'une **base d'information**. Une base d'information est un **système documentaire** permettant d'exploiter une collection de documents. La gestion concerne principalement le stockage des documents, la présentation et la recherche. Un **Système de recherche d'information** (SRI) est système permettant d'effectuer l'ensemble des tâches nécessaires à la RI. Un SRI possède trois fonctions fondamentales qui définissent le **modèle de recherche** : **représenter** le contenu des documents, **représenter** le besoin de l'utilisateur et **comparer** ces deux représentations. La **représentation** des documents et de la requête dans le système se fait à l'issue d'une phase appelée **indexation** qui consiste à choisir les termes représentatifs des documents chaque terme est associé au document dans lequel il se trouve avec éventuellement des informations additionnelles comme la fréquence d'apparition du terme dans le document. Le modèle doit mettre en correspondance les représentations des documents et la représentation du besoin de l'utilisateur exprimé sous la forme d'une requête afin de retourner à celui-ci les documents en rapport avec sa requête. Généralement, cela se fait à l'aide d'un **calcul de similarité**.

L'interrogation du fond documentaire à l'aide d'une requête nécessite la représentation de cette dernière sous une forme compatible avec celle des documents. Les fonctionnalités d'un SRI peuvent être déduites du processus global de la RI.

Notre travail est consacré à la réalisation d'un système d'indexation parallèle dont le but est d'améliorer l'efficacité en termes de temps par rapport à un système séquentiel.

Ce mémoire est organisé en quatre chapitres, le premier chapitre sera consacré aux concerts de base de la RI et l'importance des opérations qui constituent ce processus comme l'indexation, les modèle de recherche d'information et leurs évaluation avec une présentation de quelques algorithmes existants qui traitent la radicalisation tels que les algorithmes de Porter et de Lovins et de Carry. Le deuxième chapitre présentera les concepts du parallélisme. Le troisième et le quatrième chapitre seront consacré à une description du model ainsi qu'à l'implémentation avec les résultats obtenus.
Enfin, une conclusion finale avec des perspectives qui résument le travail effectué.

CHAPITRE I :
LA RECHERCHE D'INFORMATION

CHAPITRE I : LA RECHERCHE D'INFORMATION

1. Introduction

Le domaine de recherche d'information (RI) remonte au début des années 1940, après l'invention de l'ordinateur, la RI est apparue comme une réponse au besoin de gérer l'explosion de la quantité d'informations.

Ce chapitre a pour but de présenter le domaine de la RI. Dans la première partie, nous présentons les concepts de base de la RI, ensuite nous allons voir les différentes parties qui constituent l'étape d'indexation, et on s'intéressera aussi aux différents modèles de recherche.

2. La recherche d'information

2.1. Définitions

Le nom de « recherche d'information » fut donné par Calvin Mooers en 1948 pour la première fois quand il travaillait sur son mémoire de maîtrise [1]. Il existe plusieurs définitions de la recherche d'information, nous citons les trois définitions suivantes :

Définition 1 : Une des premières définitions de la RI a été donnée par Gerard Salton : « la recherche d'information est un domaine qui a pour objectif, la représentation, l'analyse, l'organisation, le stockage et l'accès à l'information » [2].

Définition 2 : La recherche d'information est une activité dont la finalité est de localiser et de délivrer des granules documentaires à un utilisateur en fonction de son besoin en informations [3].

Définition 3 : La recherche d'information est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection d'information [4].

2.2. Les concepts de base de recherche d'information

Le rôle d'un Système de Recherche d'Information (SRI) est de sélectionner les documents qui peuvent répondre au besoin en information de l'utilisateur formulé par une requête de recherche.

L'objectif principal de tout SRI est de garantir aux utilisateurs une meilleure pertinence des résultats.

L'architecture générale d'un SRI illustrée par la ***(figure 1.1)*** fait ressortir des éléments de base tels que : le document, la requête et la pertinence des documents retournés, ainsi que trois parties : l'indexation, la recherche et la reformulation de la requête.

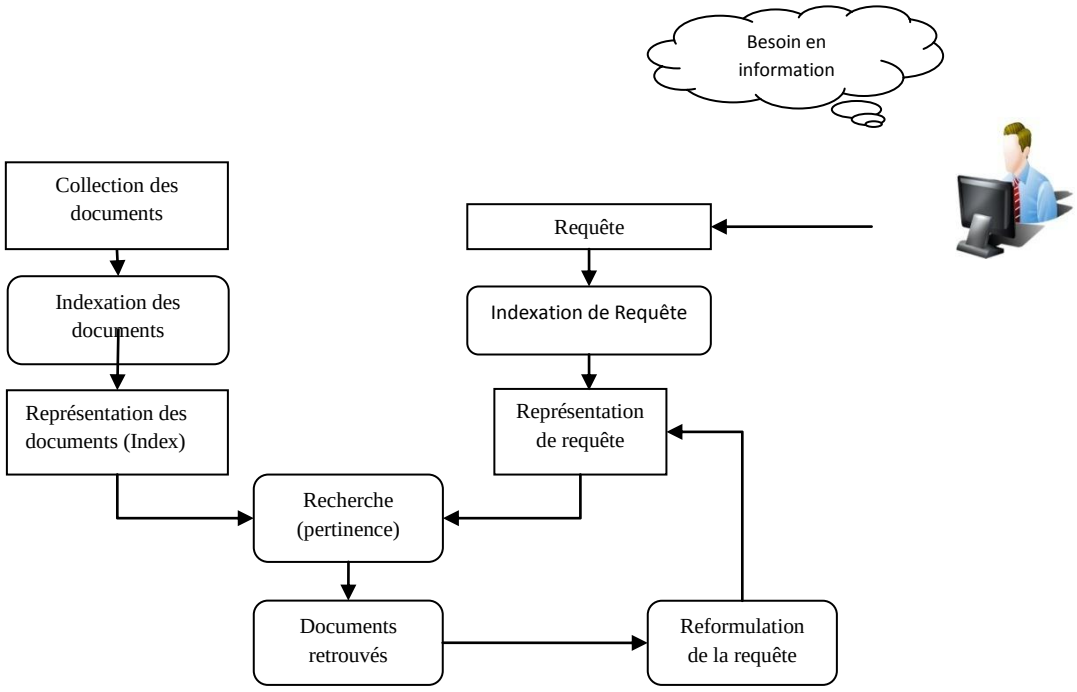


Figure 1.1 : *Processus de la recherche d'information*

2.2.1. Document et collection de documents

La collection de documents constitue l'ensemble des informations exploitables et accessibles. Elle est constituée d'un ensemble de documents.

Un document est considéré comme un support physique de l'information, qui peut être du texte, une page web, une image, une séquence vidéo, etc. Dans le cas d'un document texte on peut le représenter selon trois vues :

- **La vue sémantique (ou contenu) :** elle se concentre sur l'information véhiculée dans le document [5].
- **La vue logique** : elle définit la structure logique du document (structuration en chapitres, sections) [6].
- **La vue présentation** : elle consiste en la présentation sur un médium à deux dimensions (alignement de paragraphes, indentation, en-têtes et pieds de pages, etc.).

2.2.2. Besoin en informatique

La notion de besoin en information est souvent assimilée au besoin de l'utilisateur. Il existe trois types de besoin en information définis par Peter Ingwersen [7].

- **Besoin vérificatif :** l'utilisateur cherche à vérifier le texte avec les données connues qu'il possède déjà. Il recherche donc une donnée particulière, et sait même souvent comment y accéder.
  - Un besoin de type vérificatif est dit stable, c'est-à-dire qu'il ne change pas au cours de la recherche.
- **Besoin thématique connu** : l'utilisateur cherche à clarifier, à revoir ou à trouver de nouvelles informations dans un sujet et un domaine connu. Un besoin de ce type peut être stable ou variable ; il est très possible en effet que le besoin de l'utilisateur s'affine au cours de la recherche. Le besoin peut aussi s'exprimer de façon incomplète, c'est-à-dire que l'utilisateur n'énonce pas nécessairement tout ce qu'il sait dans sa requête mais seulement un sous-ensemble. C'est ce qu'on appelle dans la littérature le label.
- **Besoin thématique inconnu** : cette fois, l'utilisateur cherche de nouveaux concepts ou de nouvelles relations en dehors des sujets ou des domaines qui lui sont familiers. Le besoin est intrinsèquement variable et est toujours exprimé de façon incomplète
  -

2.2.3. Requête

La requête peut avoir la forme d'une expression en langue naturelle, en langage booléen ou graphique, c'est-à-dire, un ensemble de mots clés. Elle constitue l'expression du besoin en information de l'utilisateur.

2.2.4. Pertinence

Est la correspondance entre un document et une requête.

Il existe deux types de pertinence :

- **La pertinence** est souvent présentée par un score attribué par le SRI afin dévaluer l'adéquation du contenu des documents vis-à-vis de celui de la requête. Ce score est généralement évalué en fonction des poids des mots index de la requête dans le document interrogé. Ces poids représentent l'importance des mots pour le contenu d'un document [8].
- **La pertinence utilisateur**, elle est liée à l'évaluation effectuée par l'utilisateur en ce qui concerne l'information renvoyée par le système [9], [10], [11].

2.3. Le processus d'indexation

L'indexation est une étape très importante dans le processus de RI. Elle consiste à extraire des documents ou des requêtes les mots les plus discriminants encore appelés index. Cette première tâche est généralement effectuée en marge du processus de recherche car, la construction des index peut être assez longue en fonction du nombre de documents de la collection ainsi que de la taille des documents. Les index ont un caractère réducteur car tous les termes d'un document ne sont pas importants à prendre en compte pour la recherche.

L'indexation se décompose en quatre phases schématisées dans la ***(figure 1.2)***.

- L'extraction des termes du document.
- La sélection et l'élimination des termes vides.
- L'extraction des radicaux.
- La pondération des termes.

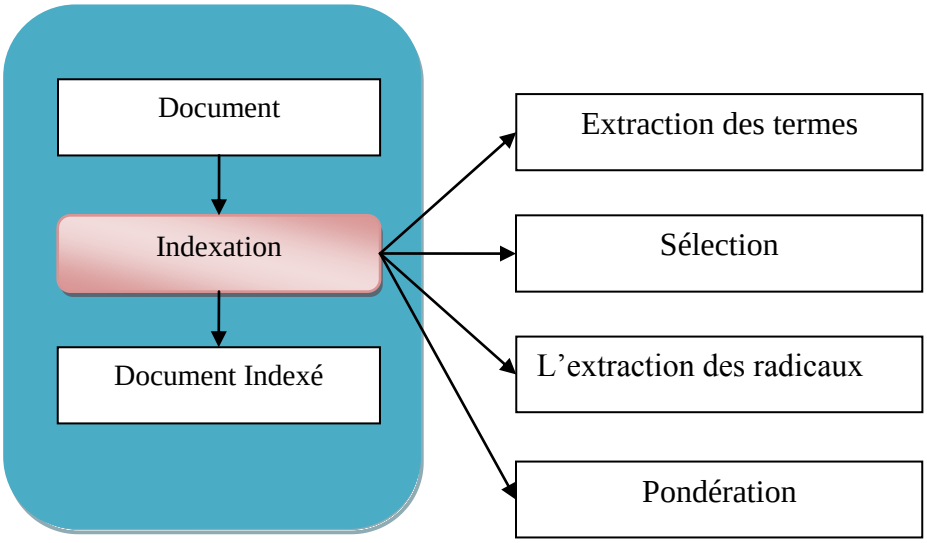


Figure 1.2 : *Indexation d'un document [12]*

Le but de l'indexation est donc de produire une représentation synthétique des documents, formés de termes, ces termes peuvent être extraits de trois manières :

- **indexation manuelle** : chaque document est analysé par un spécialiste du domaine correspondant ou par un documentaliste, L'indexation manuelle assure une meilleure précision dans les documents restitués par le SRI en réponse aux requêtes des utilisateurs [13].

- **indexation semi-automatique :** la tâche d'indexation est réalisée ici conjointement par un programme informatique et un spécialiste du domaine [14]. le choix final reste au spécialiste du domaine correspondant ou documentaliste, qui intervient souvent pour établir des relations sémantiques entre mots-clés et choisir les termes significatifs.
- **indexation automatique :** chaque document est analysé à l'aide d'un processus entièrement automatisé. Elle est réalisée par un programme informatique et elle passe par un ensemble d'étapes qui sont : l'analyse lexicale, l'élimination des mots vides de sens, la normalisation (lemmatisation ou radicalisation). Nous détaillons ces différentes étapes ci-dessous.

2.3.1. L'analyse lexicale

Cette étape est généralement indépendante de la requête de l'utilisateur, mais elle dépend du type de source de données. Elle permet de convertir un texte de document en une liste de termes.

2.3.2. L'élimination des mots vides de sens

Les mots vides de sens désignent des mots n'ayant pas de réelle signification. On dit aussi qu'ils ne sont pas porteurs de sens. On distingue deux techniques pour éliminer les mots vides de sens:

- L'utilisation d'une liste préétablie de mots vides (aussi appelée anti dictionnaire ou stop-List),
- L'élimination des mots ayant une fréquence qui dépasse un certain seuil dans la collection.

La suppression des mots vides doit être contrôlée car elle influence la qualité de la recherche.

2.3.3. La radicalisation

La radicalisation est une phase importante pour le processus d'indexation. Elle consiste à supprimer les suffixes flexionnels et dérivationnels pour réduire les différentes formes d'un mot à leur racine. La racine d'un mot correspond à la partie du mot restante une fois que l'on a supprimé son préfixe et son suffixe, à savoir son radical.

La radicalisation permet de réduire le mot à sa racine. C'est la racine qui va être utilisée comme terme d'indexation.

Exemple : Supposons que les mots "retrieval", "retrieved" "retrieving" et "retrieve" appartiennent tous à un document. Ils sont réduits à leur racine "retriev" et qui va être le terme d'indexation.

Plusieurs algorithmes ont été développés pour raffiner la tâche de radicalisation, ces algorithmes de radicalisation procèdent en deux étapes : un pas de dé-suffixation qui consiste à ôter aux mots des terminaisons prédéfinies les plus longues possibles, et un pas de recodage qui ajoute aux racines obtenues des terminaisons prédéfinies.
L'algorithme de *Lovins* fait les deux étapes séparées, mais celui de *Porter* fait les deux étapes simultanément.

2.3.3.1. L'algorithme de porter

Définitions :

L'algorithme *Porter* est l'un des plus célèbres algorithmes de radicalisation [15]. Il a été développé en 1979 par Porter au laboratoire informatique au sein de l'université du Cambridge (Angleterre). Il fait partie d'un grand projet dans le domaine de la recherche d'information. Il permet de supprimer les suffixes des mots pour obtenir une forme canonique du mot. Cet algorithme est utilisé pour la langue anglaise, mais son efficacité est limitée pour la langue française où les flexions sont plus importants et plus diverses. Il reste toutefois un algorithme fondamental couramment enseigné en TALN [16].

Principe de l'algorithme de porter :

L'algorithme de Porter fonctionne sur les mots composé leur plus bas niveaux de lettres. L'ensemble des lettres de l'alphabet latin, les voyelles et les consonnes :

- Les voyelles (v) : la liste des voyelles est (A, E, I, O, U et le Y) le "y" on le considère comme une voyelle si son prédécesseur est une consonne.
- Les consonnes (c) : sont toutes les lettres de l'alphabet latin sauf les voyelles et le 'y' si son prédécesseur est une consonne.

Quelques notations :

- V : une suite de voyelles.
- C : une suite de consonnes.
- m : un entier est appelé la mesure d'un mot.
- (VC) m : la suite "CV" se répète "m" fois.
- [C]: la suite des consonnes "C" peut existait ou pas.
- *S : signifie que le radical se termine par la lettre S
- *v* : signifie que le radical contient une voyelle
- *d : signifie que le radical se termine par deux consonnes

- *o : signifie que le radical se termine par la séquence CVC et que la dernière consonne n'est ni W, ni X et ni Y.

Un mot en anglais peut avoir l'une des 4 formes suivantes :

Forme 1 : CVCVC…C

Forme 2 : CVCVC…V

Forme 3 : VCVCV…C

Forme 4 : VCVCV…V

La formule générale est : [C] $(VC)^m$ [V].

Quelques exemples :

Si m=0: no, tree, toy, to...

T	**R**	**E**	**E**
C		**V**	
[C]		**[V]**	

m=1: in, nine, oats, trees, write, one...

m=2: troubles, private, writing, oaten, orrery …

W	**R**	**I**	**T**	**I**	**N**	**G**
C		**V**	**C**	**V**	**C**	
[C]		**m=1**		**m=2**		

Numéro d'étape		**Règles**	**Example**
Etape 1	**A**	SSES → SS IES → I SS → SS S →	caresses → caress ponies → poni caress → caress cats → cat
	B	(m>0) EED → EE (*v*) ED → (*v*) ING →	feed → feed, agreed → agree plastered → plaster, bled → bled motoring → motor, sing → sing
	C	(*v*) Y → I	happy → happi, sky → sky

Etape 2	(m>0)ational ->Ate (m >0)tional -> tion (m >0)enci ->ence (m >0)anci -> ance (m >0)izer ->ize (m >0)abli ->able (m >0)alli ->al (m >0)entli ->ent (m >0)eli -> e (m >0)ousli ->ous (m >0)ization -> ize (m >0)ation ->ate (m >0)ator -> ate (m >0)alism ->al (m >0)iveness ->ive (m >0)fulness -> ful (m >0)ousness ->ous (m >0)aliti -> al (m >0)iviti ->ive (m >0)biliti ->ble	relational ->relate Conditional -> Condition valenci -> valence hesitanci ->hesitance digitizer ->digitize conformabli -> conformable radicalli ->radical differentli ->different vileli ->vile analogousli -> analogous vietnamization -> vietnamize predication ->predicate operator -> operate feudalism ->feudal decisiveness ->decisive hopefulness ->hopeful callousness ->callous formaliti ->formal sensitiviti ->sensitive sensibiliti ->sensible
Etape 3	(m >0)icate -> ic (m >0)ative -> (m >0)alize ->al (m >0)iciti ->ic (m >0)ical ->ic (m >0)ful -> (m >0)ness ->	triplicate -> triplic formative ->form formalize ->formal electriciti ->electric electrical ->electric hopeful ->hope goodness ->good
Etape 4	(m >1)al -> (m >1)ance -> (m >1)ence -> (m >1)er -> (m >1)ic -> (m >1)able -> (m >1)ible -> (m >1)ant -> (m >1)ement -> (m >1)ment -> (m >1)ent -> (m >1 et (* s ou * t))ion -> (m >1)ou -> (m >1)ism -> (m >1)ate -> (m >1)iti -> (m >1)ous -> (m >1)ive -> (m >1)ize ->	revival -> reviv allowance->allow inference ->infer airliner -> airlin gyroscopic -> gyroscop adjustable ->adjust defensible -> defens irritant -> irrit replacement ->replac adjustment -> adjust dependent -> depend Adoption ->adopt homologou ->homolog communism -> commun activate -> activ angulariti -> angular homologous -> homolog effective -> effect bowdlerize -> bowdler

Etape 5	(m>1) E → (m=1 et not *o) E → (m>1 et *d et *L) → lettre non doublée	probate → probat, rate → rate cease → ceas controll → control, roll → roll

Tableau 1.1 : *Algorithme de Porter*

Exemple : En appliquant les différentes étapes de l'algorithme de Porter on aura :

Généralisations

- étape 1: Generalization**s** → Generalization
- étape 2: Generaliz**ation**→ Generaliz**e**
- étape 3: General**ize**→ General
- étape 4: Gener**al**→ Gener

Oscillators

- étape 1: Oscillators → Oscillator
- étape 2: Oscillator → Oscillate
- étape 4: Oscillate → Oscill
- étape 5: Oscil**l**→ Oscil

2.3.3.2. L'algorithme de Carry

L'algorithme de Carry a été proposé pour l'étude de la morphologie de français, c'est la version améliorée de l'algorithme de Porter avec en plus les suffixe du français [17].

Tout comme celui de Porter, l'algorithme de Carry se déroule en diverse étapes par lesquelles les mots à traiter passent successivement. Selon les règles, quand l'analyseur reconnaît un suffixe de la liste, soit il le supprime, soit il le transforme. C'est ici aussi le suffixe le plus long qui détermine la règle à appliquer.

2.3.3.3. Algorithme de Lovins

Le premier algorithme de racinisation a été écrit par Julie Beth Lovins en 1968 [18]. Ce document a été remarquable par sa date de début et avait une grande influence sur le travail plus tard dans ce domaine. L'algorithme de Lovins a 294 terminaisons, 29 conditions et 35 règles de transformation et où chaque terminaison est associée à l'une des conditions. Dans la première étape, si la plus longue terminaison trouvée satisfait sa condition qui lui est associée, cette terminaison sera éliminée. Dans la deuxième étape, les 35 règles sont appliquées pour transformer la terminaison. La seconde étape est effectuée si une terminaison est supprimée dans la première étape ou non.

2.3.4. La pondération

L'objectif principal de la pondération est d'assigner aux termes d'index des poids sensés traduire leur importance dans les documents où ils apparaissent.

Le calcul du poids des termes au sein des nœuds feuilles n'est pas un problème trivial. Ce poids doit modéliser l'importance du terme dans le nœud feuille, mais aussi au sein du document et de la collection.

Le calcul de $\boldsymbol{wij}$ dépend du modèle de pondération considéré :

$$\boldsymbol{wij} = \frac{tfij}{dfj} = tfij * \frac{1}{dfj} = tfij * idfj$$

Où :

- *tfij* est la fréquence d'occurrences du terme *tj* dans le document *di*.
- *dfj* est la fréquence documentaire du terme *tj* (i.e. la proportion de documents de la collection qui contiennent *tj*) et *idfj* sa fréquence documentaire inverse.

La mesure *tf* idf* est une bonne approximation de l'importance d'un terme dans un document, particulièrement dans des corpus de documents de tailles intermédiaires.

2.3.5. Le résultat d'une indexation

Le résultat d'une indexation est donc un ensemble de termes qui peuvent être soit un mot, soit une racine de mot soit un terme composé si on possède un mécanisme pour reconnaître des termes composés.

2.4. Les modèles de recherche d'information

Le modèle de RI permet de donner une interprétation des termes choisis pour représenter le contenu d'un document. Plusieurs modèles de RI ont été proposés dans la littérature, on distingue trois principales catégories de modèles ***(Figure 1.3)*** : modèles booléens, modèles vectoriels et modèles probabilistes [20].

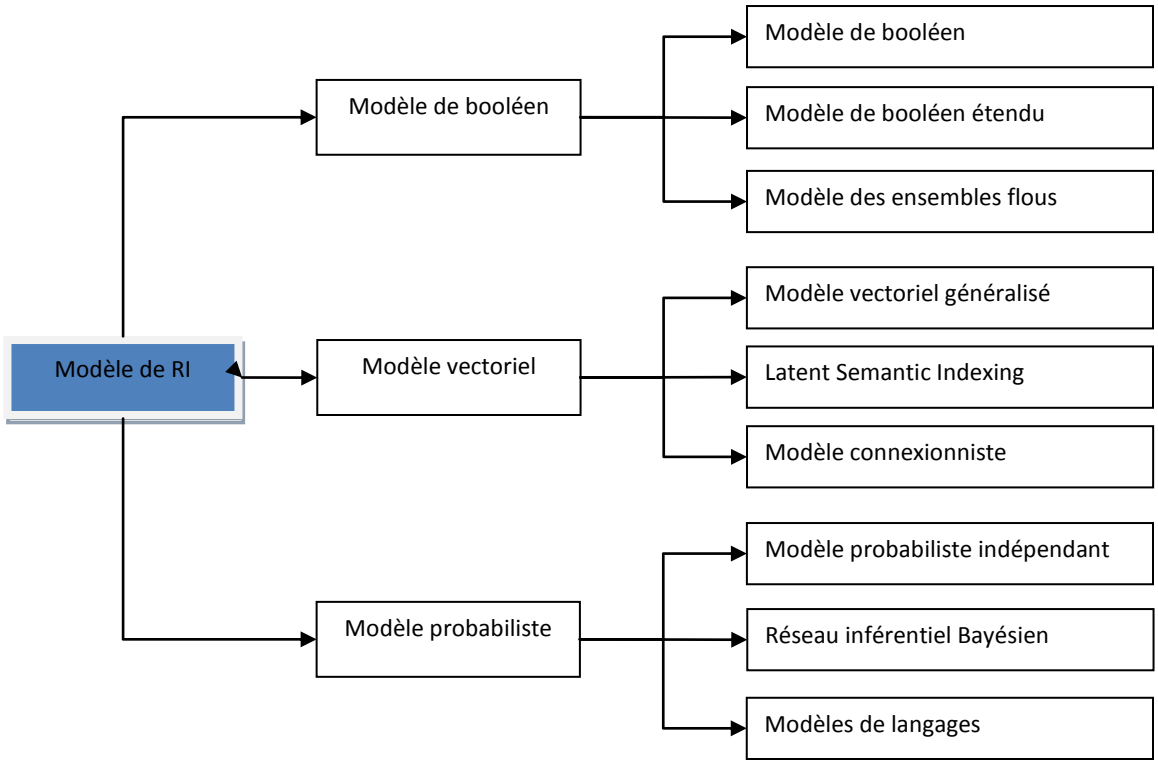


Figure 1. 3 *: Différents modèles de la recherche d'information*

Le modèle remplit les deux rôles suivants:

- Créer une représentation interne pour un document ou pour une requête basée sur ces termes.
- Définir une méthode de comparaison entre une représentation de document et une représentation de requête afin de déterminer leur degré de correspondance.

Le modèle joue un rôle central dans la RI. C'est le modèle qui détermine le comportement clé d'un système de RI.

2.4.1. Les modèles booléens

Les premiers des modèles sont basés sur la théorie des ensembles et l'algèbre de Boole [21]. Les documents sont représentés par une conjonction des termes qui constituent leur contenu (par exemple d = $\{t1 \Lambda t2 \Lambda, \ldots, \Lambda tk\}$) et les requêtes sont formulées à l'aide d'expressions logiques Et (Λ), Ou(Λ), Non (¬). (Par exemple: q = (t1 Λ t2) Λ (t3 Λ ¬t4).

La figure suivante montre différents ensembles restitués (parties de disques grisés) pour différentes requêtes :

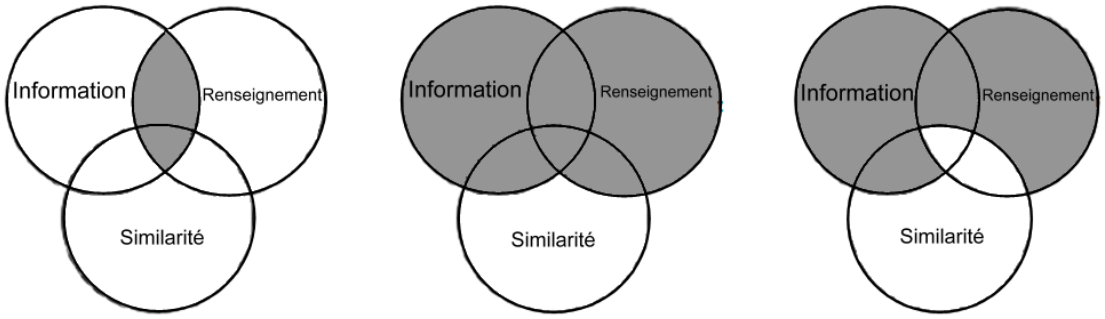


***Figure 1.4** : Requêtes booléennes sous forme de diagramme de Venn [22].*

Un document est jugé pertinent si et seulement si son contenu respecte la formule logique de la requête. Des poids binaires (0 et 1) sont utilisés pour la pondération d'un terme. Ainsi, les termes de la requête sont soit présents, soit absents dans le document.

La correspondance R (d, q) entre une requête et un document est déterminée de la façon suivante :

R (d, ti) = 1 si ti ∈ d; 0 sinon.

R (d, q1 ∧ q2) = 1 si R (d, q1) = 1 et R (d, q2)= 1; 0 sinon.

R (d, q1 ∨ q2) = 1 si R (d, q1) = 1 ou R (d, q2) = 1; 0 sinon.

R (d, ¬q1) = 1 si R (d, q1) = 0; 0 sinon.

Les principaux problèmes liés à ce modèle sont:

- La sélection d'un document est basée sur une décision binaire
- Pas d'ordre pour les documents sélectionnés
- Formulation de la requête difficile pas toujours évidente pour beaucoup d'utilisateurs
- Problème de collections volumineuses : le nombre de documents retournés peut être considérable

2.4.2. Les modèles vectoriels

Le modèle vectoriel est un modèle algébrique, de base a été introduit par Salton, concrétisé dans le cadre du système SMART [23].

Les modèles vectoriels ont été créés pour compenser la limitation de la pondération binaire des modèles booléens. Ils sont basés sur l'attribution de poids non binaires aux termes indexés dans les documents et aux termes des requêtes.

Ce modèle préconise la représentation des requêtes utilisateurs et documents sous forme de vecteurs dans l'espace **T** de termes d'indexation de dimension N, **T**= $\{t_1, t_2, \dots, t_j, \dots, t_n\}$.

Un document $\boldsymbol{d_i}$ est représenté par un vecteur $\boldsymbol{d_i}\{w_{i1}, w_{i2}, \dots, w_{ij}, \dots, w_{in}\}$. Une requête **q** par un vecteur **q** $\{w_{q1}, w_{q2}, \dots, w_{qj}, \dots, w_{qn}\}$.

Où $\boldsymbol{w_{ij}}$ est le poids du terme $\boldsymbol{t_j}$ dans le document $\boldsymbol{d_i}$ **(**respectivement dans la requête **q**). Ce poids peut être soit une forme de ***tf*idf*** soit un poids attribué manuellement par l'utilisateur.

Plusieurs mesures de similarité ont été définies [24], dont les plus courantes sont décrites dans le ***Tableau 1.1*** ci-dessous.

Mesures	**Formules**
Le produit scalaire	RSV (q,d_i)=$\sum_{j=1}^{n}$ Wqj $*$ Wij
La mesure de cosinus	RSV (q,d_i)=$\frac{\sum_{j=1}^{n} \text{Wqj} * \text{Wij}}{\left(\sum_{j=1}^{n} \text{Wqj}\right)^{½} * \left(\sum_{j=1}^{n} \text{Wij}\right)^{½}}$
La mesure de Dice	RSV (q,d_i)= $2 * \frac{\sum_{j=1}^{n} \text{Wqj} * \text{Wij}}{\left(\sum_{j=1}^{n} \text{Wqj}^{2}\right) + \left(\sum_{j=1}^{n} \text{Wij}^{2}\right)}$
La mesure de Jaccard	RSV (q,d_i)=$\frac{\sum_{j=1}^{n} \text{Wqj} * \text{Wij}}{\left(\sum_{j=1}^{n} \text{Wqj}^{2}\right) + \left(\sum_{j=1}^{n} \text{Wij}^{2}\right) - \sum_{j=1}^{n} \text{Wqj} * \text{Wij}}$
La mesure de Superposition	RSV (q,d_i)=$\frac{\sum_{j=1}^{n} \text{Wqj} * \text{Wij}}{\min\left(\left(\sum_{j=1}^{n} \text{Wqj}^{2}\right), \left(\sum_{j=1}^{n} \text{Wij}^{2}\right)\right)}$

Tableau 1.2*: Les mesures de similarité utilisées dans le modèle vectoriel*

Les avantages du modèle vectoriel :

- La pondération améliore les résultats de recherche.
- La mesure de similarité permet d'ordonner les documents selon leur pertinence vis à vis de la requête.

Les inconvénients du modèle vectoriel :

- Difficulté d'aller plus avant dans le cadre vectoriel (modèle relativement simple).
- La représentation vectorielle suppose l'indépendance entre termes.

2.4.3. Les modèles probabilistes

Les modèles probabilistes ont également été proposés pour compléter les modèles classiques de la RI, ils s'appuient sur la théorie des probabilités [25]. Ils trient les documents selon leur probabilité de pertinence vis-à-vis d'une requête. La fonction de classement (tri) de ce modèle est exprimée ainsi :

$$RSV(q,d) = \frac{p\left(\frac{per}{q.d_i}\right)}{p\left(\frac{Nper}{q.d_i}\right)}$$

L'idée de base de cette fonction est de sélectionner les documents ayant à la fois une forte probabilité d'être pertinents et une faible probabilité d'être non pertinents à la requête.

Où $p\left(\frac{per}{q.d_i}\right)etp\left(\frac{Nper}{q.d_i}\right)$: la probabilité qu'un document d_i soit pertinent (per) vis-à-vis de la requête q (respectivement non pertinent (Nper).

Avantages du modèle probabiliste :

- Apprentissage du besoin d'information
- Une notion claire et théoriquement fondée du degré de pertinence

Inconvénients du modèle probabiliste :

- Le modèle considère que tous les termes sont indépendants
- Pas de langage de requête
- Problème des probabilités initiales

2.5. La reformulation de la requête

La reformulation de la requête permet d'enrichir une requête initiale en fonction de jugements de pertinence afin d'exprimer d'avantage les besoins de l'utilisateur. Elle est souvent opérée par ajout et/ou réévaluation des poids des termes de la requête initiale.

Les techniques de reformulation de la requête peuvent être classées en tenant compte de plusieurs paramètres [26] :

- Les sources de données utilisées pour l'expansion de requête.
- La méthode de sélection des termes d'expansion : la relation de cooccurrence, les mesures d'information, les techniques de classification, etc.
- La sélection des termes d'expansion en considérant chaque terme de la requête individuellement, ou la requête dans son ensemble.

- La représentation de la requête (document) comme un ensemble de mots simples (sac de mots) ou une représentation prenant en compte les relations de proximité entre termes ;
- Le type de terme d'expansion (mot simple ou mot composé).
- L'intervention de l'utilisateur dans le processus d'expansion de la requête (automatique, manuelle, semi-automatique).

Il existe trois méthodes de reformulation de requêtes :

2.5.1. La reformulation manuelle

Cette approche est associée aux systèmes de recherche booléens. Elle consiste à présenter à l'utilisateur une liste de documents jugés pertinents en réponse à la requête initiale. C'est à l'utilisateur de sélectionner à partir des documents pertinents ceux dont lesquels le système va extraire les termes à rajouter à la requête initiale dans le but d'effectuer une nouvelle recherche.

2.5.2. La reformulation semi-automatique

Cette technique nécessite l'intervention de l'utilisateur qui doit identifier et sélectionner les documents pertinents et les documents non pertinents. A l'inverse de la reformulation automatique, ici, ce sont le système et l'utilisateur qui sont, ensemble, responsables de la détermination et du choix des termes candidats à la reformulation

2.5.3. La reformulation automatique

L'extension de la requête est faite sans intervention de l'utilisateur grâce à l'utilisation d'un thésaurus contenant des informations de type linguistique (équivalence, association, hiérarchie) et statistique (pondération des termes).

Le problème avec la reformulation automatique est l'estimation des « bons » termes qui peuvent conduire effectivement à une amélioration du processus de recherche car l'introduction des termes inappropriés peut entraîner un silence ou au contraire augmenter un bruit.

2.6. Appariement document-requête

La correspondance entre les termes de la requête d'un utilisateur et ceux des documents s'effectue au niveau de l'appariement document-requête, cette étape sert à renvoyer une liste de documents ordonnées selon un degré de pertinence, ce dernier est calculé à partir d'une fonction notée RSV (Q, D).

Il a été défini formellement par un quadruplet (D, Q, F, R (q, d)) [19] où :

- D est l'ensemble des documents de la collection,
- Q est l'ensemble des requêtes des utilisateurs,
- F est le schéma du modèle théorique de représentation des documents et des requêtes,
- RSV (q, d) est la fonction de pertinence du document d à la requête q.

3. Conclusion

Dans ce chapitre nous avons passé en revue les principaux concepts de la recherche d'information.

Le second chapitre sera consacré aux concepts de base du parallélisme sur lesquels se base notre système.

.

CHAPITRE II :

LE PARALLELISME

CHAPITRE II : LE PARALLELISME

1. Introduction

Le parallélisme en informatique consiste à résoudre des problèmes scientifiques le plus rapidement possible par apport à la résolution séquentielle, le principe de base du parallélisme est d'utiliser plusieurs processeurs qui fonctionnent en même temps dont le but est d'augmenter la puissance et d'accélérer de calcul pour la résolution d'une même tâche pour un problème donné.

2. Les concepts de base de parallélisme

2.1. Terminologie

- **Tâche *:*** en informatique, une tâche ou processus est une unité de travail exécutée sur un ordinateur et défini par un ensemble d'instructions.
- **Tâche parallèle :** c'est un processus qui peut s'exécuter sur plusieurs processeurs à condition d'obtenir le même résultat pour une exécution séquentielle.
- **Ordinateur parallèle :** c'est une machine qui utilise plusieurs processeurs en même temps pour résoudre un problème.
- **Exécution parallèle :** c'est l'exécution de plusieurs instructions en même temps.
- **L'accélération:** l'accélération d'un programme parallèle est le rapport entre le temps d'exécution séquentiel et le temps d'exécution parallèle résolvant le même problème sur une machine parallèle donnée, elle est définie par: **Sp** = T_s/T_p Généralement on a : s<**Sp**< p

  $\mathbf{T_s}$ temps d'exécution séquentiel.

  $\mathbf{T_p}$: temps d'exécution parallèle sur **p** processeurs.
- **L'efficacité :** traduit le taux d'utilisation des **p** processeurs de la machine parallèle, elle est définie par: **Ep**= Sp/p.
- **Système distribué :** (ou réparti) c'est un système composé de plusieurs processeurs impliqués dans la résolution d'un ou plusieurs problèmes**.**
- **Mémoire partagée :** dans cette machine, il y a plusieurs processus qui ont un accès direct à une mémoire commune. La communication entre les tâches est réalisée par les opérations de lecture et d'écriture sur la mémoire partagée ***(figure 2.1)***.

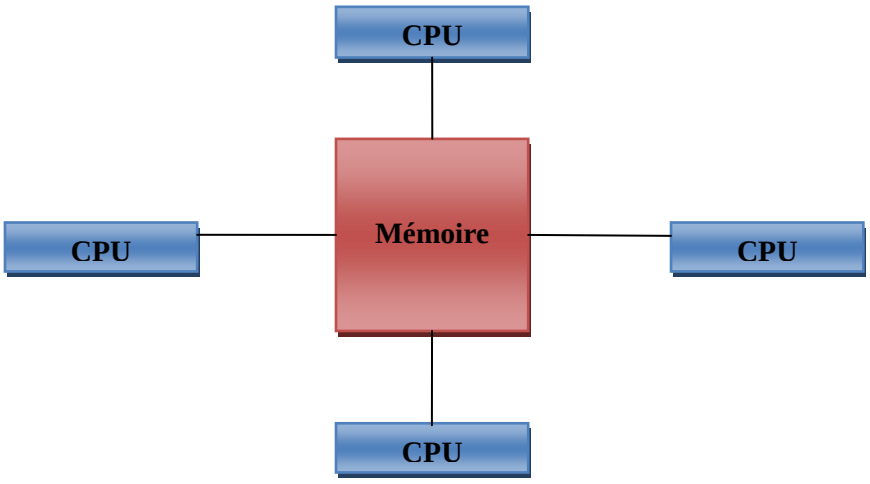


Figure 2.1 : *Machine à mémoire partagé*

- **Mémoire distribuée :** dans cette machine, chaque processeur a sa propre mémoire locale. La communication entre les processeurs ***via un réseau d'interconnexion (figure 2.2)***.

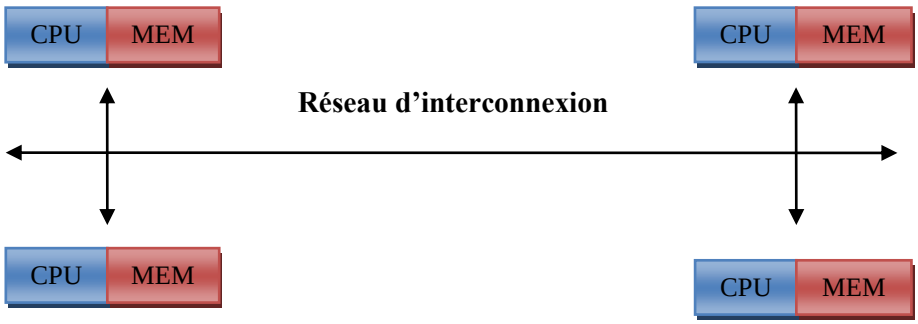


Figure 2.2 : *Machine à mémoire distribuée*

- **Machine à mémoire hybride :** Ce type de machine il est mixe entre les deux types précédents (mémoire partagée et mémoire distribuée ***(figure 2.3)***.

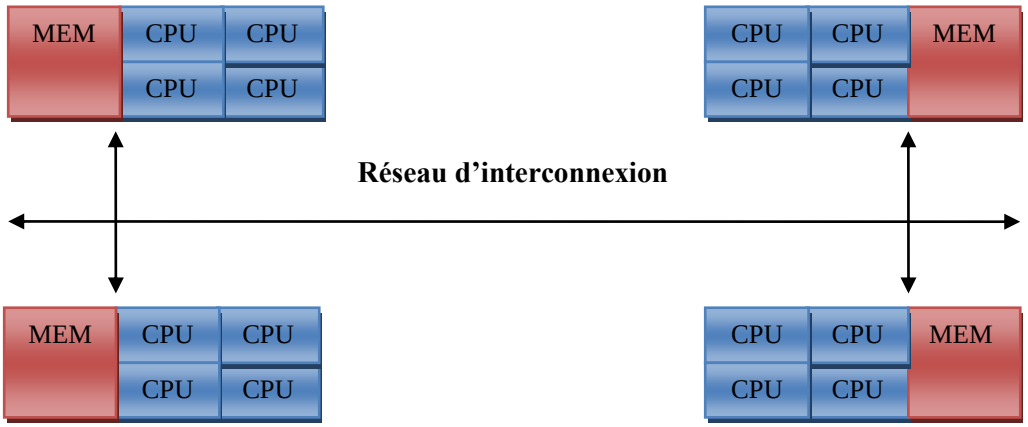


***Figure 2.3* :** *Machine à mémoire hybride*

2.2. Le but du parallélisme

a) Pourquoi le parallélisme ?

Son objectif est d'obtenir de meilleur résultat en termes de performance par rapport à un algorithme séquentiel.

b) Bénéfice de la parallélisassions

- Exécution plus rapide du programme
- Résolution de problèmes plus gros
- Diminution du temps d'exécution

2.3. Quelques domaines du parallélisme

Ils existent plusieurs domaines où le parallélisme est appliqué en informatique ; *les nids de boucles, les graphes,* ou encore *les réseaux de tri.*

Parmi les réseaux de tri qu'on peut trouver on a ;

4.3.1. Tri par fusion de BATCHER

Utilise récursivement un sous-réseau permettant de calculer la fusion des deux listes avec les conditions suivantes : Les deux suites doivent avoir une taille de puissance de deux et les deux suites doivent être triées.

Soit une liste des entiers nommée « Liste », on note **TRI**(Liste) la liste à trier, si la suite est déjà triée on note **TRIEE** (Liste). Aussi on note **FUSION** l'opérateur de fusion de deux suites triées, par exemple : **TRIEE**(Liste1), **TRIEE**(Liste2) **FUSION** ((Liste1), (Liste2)).

Description du réseau de fusion :

Supposons que les deux suites sont de taille deux à la puissance de **m.**

Pour **m=0** il suffit d'un seul comparateur.

Pour **m=1** on peut utiliser trois comparateurs.

Pour **m=n** on utilise deux copies de réseau FUSION (m-1). La première copie FUSION (m-1) fusionne les éléments d'indice pair et la seconde fusionne les éléments d'indice impair avec les hypothèses suivantes :

- COPIE1= FUSION ((les impaires de Liste1), (les impaires deListe2)).
- COPIE2= FUSION ((les paires de Liste1), (les paires deListe2)).
- TRIEE (I^{er} de copie 1, Min (IIème de copie1, I^{er} copie2), Max (IIème de copie1, I^{er} copie2),…, nième de copie 2).

Exemple 1 : on a deux listes triées **List-1** (a1, a2) **et List-2** (b1, b2)

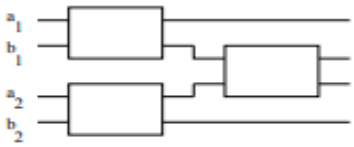


Figure 2.4 : *Tri de deux listes de taille 1*

Exemple 2 : on a deux listes triées **List-1** (a1, a2, a3, a4) **et List-2** (b1, b2, b3, b4)

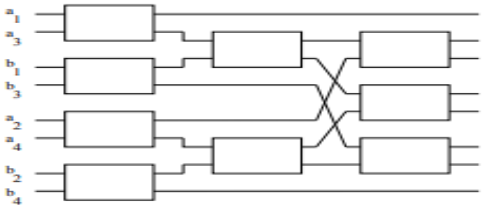


Figure 2.5:*Tri de deux listes de taille 2*

4.3.2. Tri par transposition pair-impair

Le réseau est formé d'une succession de lignes de comparateurs. Plus précisément, pour trier une liste de taille **n** éléments il doit faire :

a) **Si la taille de la liste des éléments est paire :** il faut faire **n/2** copies de réseau formé de deux lignes, La première ligne consiste en **n/2** comparateurs qui comparer entre deux élément de la liste en commençant par le premier élément, la deuxième ligne consiste en **n/2-1**comparateurs qui comparer entre deux élément de la liste en commençant par le deuxième élément donc il reste le premier élément et le dernier sans comparaison.

b) Si la taille de la liste des éléments est impaire : le même principe de la taille de la liste des éléments est pair sauf que le nombre de copie égalé **(n+1)/2** et chaque deuxième ligne de ces copies contient **n/2** comparateurs et la dernière copie contient seulement la première ligne.

La ***Figure 2.10*** représente deux exemple de construction du réseau de transposition pair impair, le premier exemple la taille de la liste égale 8 donc il faut faire 8/2 copies de réseau, le seconde la taille de la liste des éléments égale 7 donc il faut faire (7+1)/2 copies de réseau et la dernière copie contient seulement la première ligne.

Exemple :

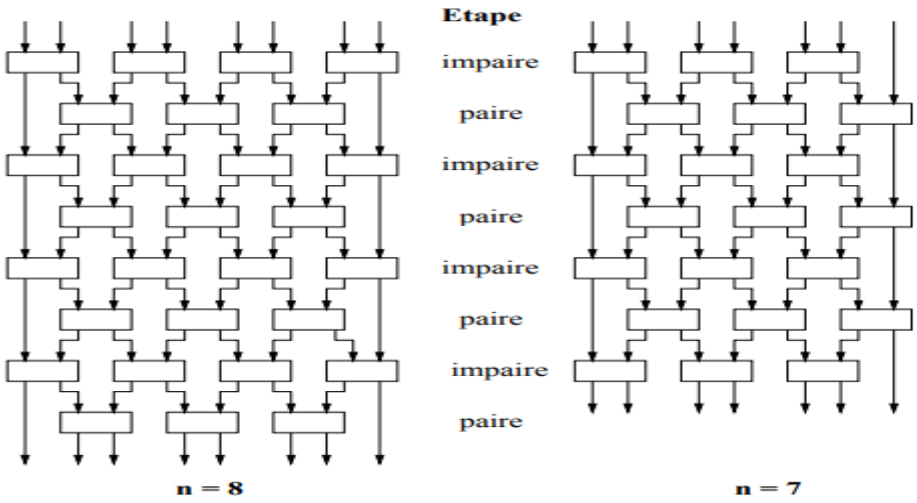


Figure 2.6 : construction du réseau de transposition pair impair

4.3.3. Tri pair-impair sur réseau linéaire de processeurs

Il faut que le nombre de processeurs **p** soit inferieur ou égale au nombre d'éléments **n** de la liste**.** Supposons **n** divisible par **p** : chaque processeur dispose au début d'une sous-suite de taille **n**/**p**, puis chaque processeur tri sa sous-suite on appel cette étape le tri locale.

Ensuite les processeurs d'indice impair communiquent avec leur voisin de gauche on fusionnant leurs éléments avant de les triés. La même opération sera répétée avec les processeurs d'indice pair communiquent avec leur voisin de droite, et ainsi de suite jusqu'à les n/p étapes (***Figure 2.11***).

Exemple :

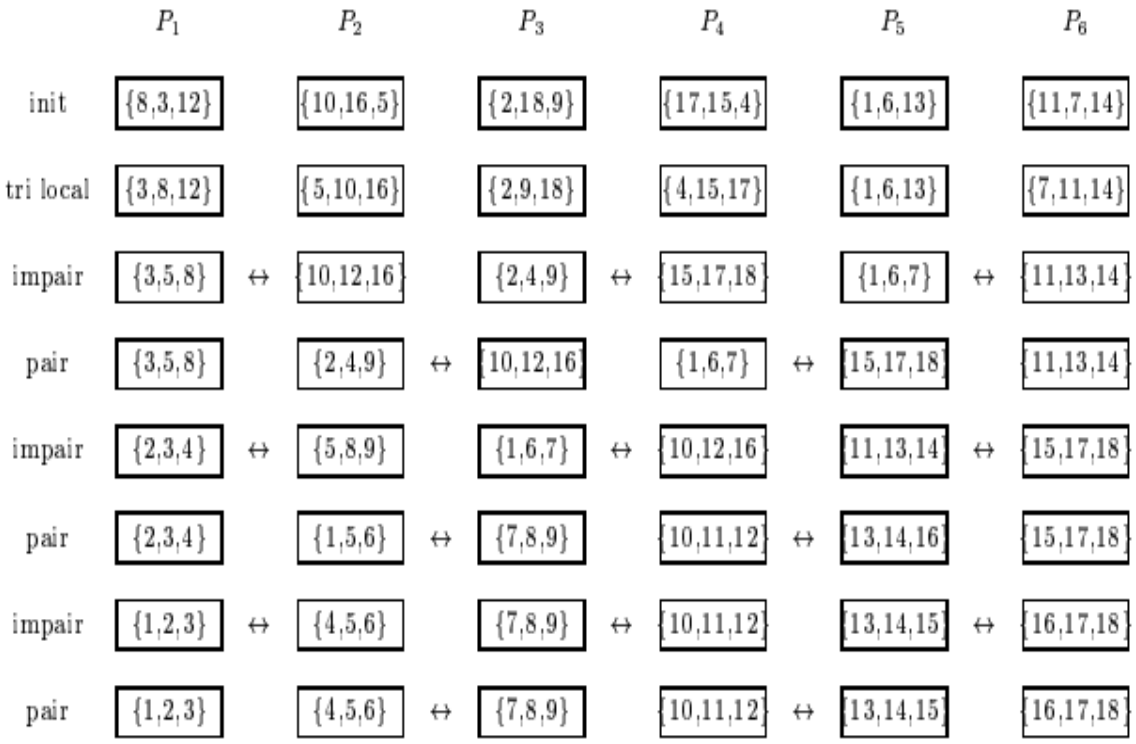


Figure 2.7 : Tri pair-impair sur réseau linéaire de processeurs

3. Architectures parallèles

Existent différentes classifications qui ont été proposées afin de caractériser les types des architectures existantes. La classification la plus connue est celle de Flynn [27]. Elle caractérise les machines selon leur mode de fonctionnement, c'est-à-dire la multiplicité des flots de contrôle et de données. Il en découle quatre catégories qui sont ci-dessous:

- **SISD : (Single Instruction Single Data):** ne traite qu'une seule donnée par instruction. (Séquentiel).
- **SIMD (Single Instruction Multiple Data):** une instruction est exécutée sur plusieurs données.
- **MISD (Multiple Instruction Single Data):** les instructions sont exécutées plusieurs fois sur une donnée unique.
- **MIMD (Multiple Instruction Multiple Data):** plusieurs processeurs traitent, en parallèle, plusieurs données.

Flot d'instructions : séquence d'instructions exécutées par la machine.

Flot de données : séquence des données appelées par le flot d'instructions.

Le ***(tableau 2.1)*** illustre la classification de Flynn.

		Flot de Données	
		Simple	Multiple
Flot d'instructions	Simple	**SISD (séquentiel)**	**SIMD**
	Multiple	**MISD**	**MIMD**

***Tableau 2.1**: La classification de Flynn.*

4. Modèles de programmation parallèle

Un modèle de programmation ou de calcul parallèle permet d'exprimer la façon dont une application parallèle sera programmée. Plusieurs modèles existent pour cette programmation. A chaque architecture de machine parallèle correspond à un modèle de programmation parallèle.

4.1. Le modèle à mémoire partagée

Dans ce modèle, il y a plusieurs processeurs avec des horloges indépendantes et une seule mémoire commune à tous les processeurs.

La ***figure 2.4*** représente une architecture qui contient plusieurs processeurs communicant entres eux via une mémoire commune.

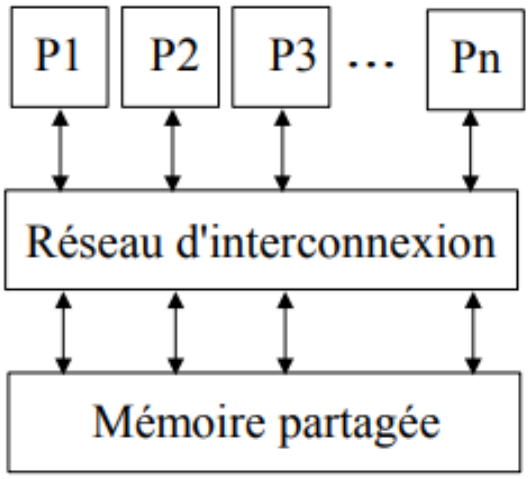


***Figure 2.8:** modèle à mémoire partagée*

Trois classes de la classification des systèmes de mémoire partagée sont détaillées ci-dessous :

4.1.1. UMA (Uniform Memory Access) :

Dans le système UMA, la mémoire partagée est accessible à tous les processeurs via un réseau d'interconnexion de la même manière à un seul processeur accède à sa mémoire. Tous les processeurs ont un accès égal à n'importe quel emplacement de mémoire. Le réseau d'interconnexion utilisé dans l'UMA peut être un ou plusieurs bus. L'accès à la mémoire partagée étant équilibré, ces systèmes sont également appelés SMP (Symmetric MultiProcessor) [28].*La* ***figure 2.5*** représente une architecture UMA (Uniform Memory Access).

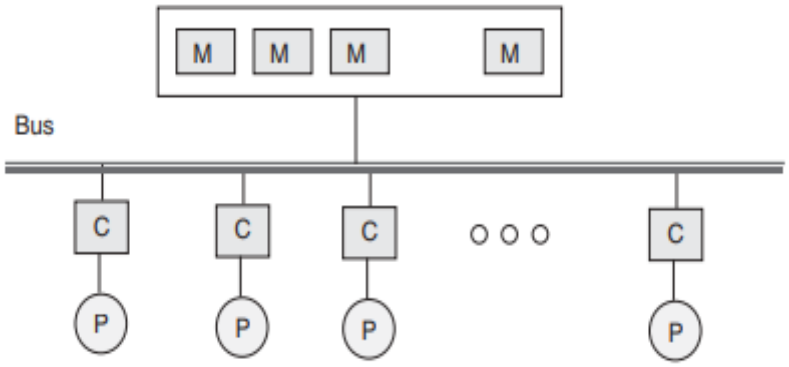


Figure 2.9: *Architecture UMA (Uniform Memory Access)*

4.1.2. NUMA (Non Uniform Memory Access):

Dans le système NUMA, une partie de la mémoire partagée est associée à chaque processeur. La mémoire a un seul espace d'adressage. Par conséquent, tout processeur peut accéder à tout emplacement de la mémoire directement en utilisant son adresse réelle [29]. *La* ***figure 2.6*** représente une architecture NUMA (Non Uniform Memory Access).

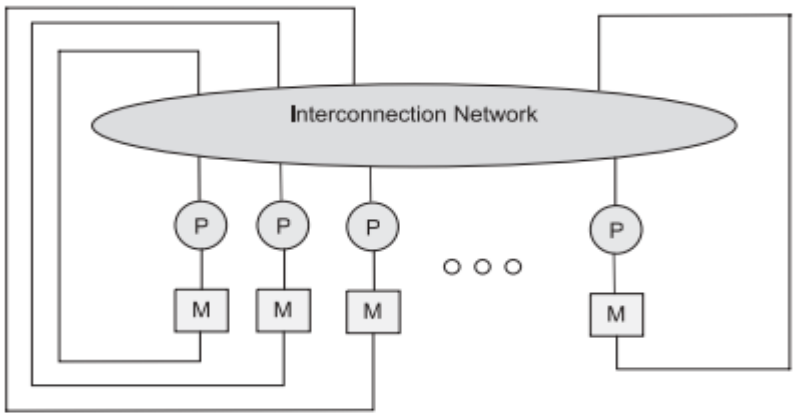


Figure 2.10: *Architecture NUMA (Non Uniform Memory Access)*

4.1.3. COMA (Cache-Only Memory Architecture)

Similaire au système NUMA, chaque processeur accède à une partie de la mémoire partagée dans le COMA. Cependant, la mémoire partagée consiste en une mémoire cache. Le système COMA exige que les données soient migrées vers le processeur qui les demande. Il n'y a pas hiérarchie de la mémoire et l'espace d'adresse est constitué de tous les caches. Il y a un répertoire de cache qui facilite l'accès aux caches distantes [30]. *La* ***figure 2.7*** montre l'organisation de COMA.

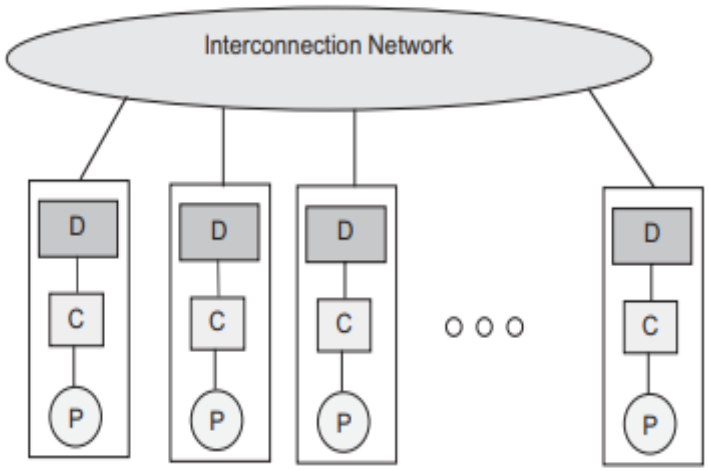


Figure 2.11 : Système de mémoire partagée COMA

Avantage :

- Simplicité d'utilisation.
- Le partage des données entre les tâches est rapide.
- On peut ajouter des processeurs quand on a besoin.
- Parallélisassions de haut niveau.

Inconvénient:

- Coûts très élevés ainsi que la complexité de réalisation.
- Limitation du nombre de processeurs (conflit d'accès au niveau matériel).
- La bande passante du réseau est le facteur limitant de ces architectures.

4.2. Le modèle à mémoire distribuée

Dans ce modèle, il y a un réseau d'interconnexion pour communiquer entre les mémoires des différents processeurs et une mémoire propre locale à chaque processeur qui exécute des

instructions identiques ou non, sur des données identiques ou non, le parallélisme par échange de messages [31].

La ***figure 2.8*** représente une architecture qui contient plusieurs processeurs liés avec un réseau de communication où chaque processeur à sa propre mémoire locale**.**

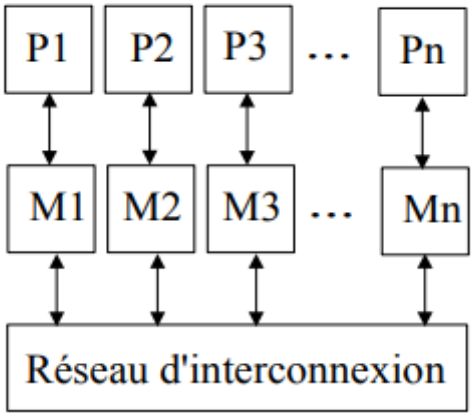


Figure 2.12: *modèle à mémoire distribué*

Avantage :

- Nombre de processeurs non limité (grande puissance de calcul).
- Mise en réseau d'un certain nombre de machines.
- Partage transparent des ressources.
- Chaque processeur a un accès rapide à sa propre mémoire.
- Cout réduit et facile à construire.

Inconvénient :

- Plus difficile à programmer que les machines à mémoire partagée.
- Les performances seront parfois au prix d'un réseau d'interconnexion couteux.

5. Conclusion

Dans ce chapitre nous avons passé en revue les différents concepts de bases ainsi que les architectures parallèles et les différents modèles de programmation parallèle nécessaires pour le parallélisme dans ce travail.

CHAPITRE III :

CONCEPTION DU SYSTEME

CHAPITRE III : CONCEPTION ET IMPLEMENTATION

1. Introduction

Dans notre travail on va construire un système séquentiel pour l'indexation d'une base de documents qui contient des textes en anglais. Le système en question effectuera les étapes suivantes ; l'analyse lexicale, supprimer les mots vides de sens, et utilise l'algorithme de *Porter* pour obtenir les radicaux des mots.

Ensuite on va construire un deuxième système qui lui est basé sur les techniques du parallélisme et comparer les résultats des deux systèmes en fonction du temps d'exécution sur la même base des documents pour prouver l'efficacité du système parallèle.

2. Architecture du système

Dans le système séquentiel on va appliquer les différentes étapes d'indexation sauf l'étape de la pondération sur une base de documents textuels numériques rédigés dans la langue anglaise, pour obtenir les radicaux des mots.

- **L'analyse lexicale :** dans cette étape on va éliminer la ponctuation, la casse et la mise en page d'un document textuel puis on va convertir un texte de document en une liste de termes.
- **La sélection :** dans cette étape on va supprimer les mots vides de la langue anglaise. Pour ce faire, on utilise un anti-dictionnaire qui contient une liste des mots vides à supprimer de l'indexation du document pour qu'il nous reste seulement les mots ayant de réelles significations.
- **L'extraction des radicaux :** la radicalisation permet de réduire le mot à sa racine. Dans cette étape il y a plusieurs algorithmes qui ont été développés, nous avons choisi l'algorithme de *Porter* pour son efficacité prouvé, ainsi que le faite qu'il est appliqué à des documents en langue anglaise, (cet algorithme est détaillé dans le chapitre 1).

Par la suite nous allons paralléliser notre système.

Enfin, nous allons comparer les résultats des deux systèmes en fonction du temps d'exécution sur la même base de documents.

La figure 3.1 Représente l'architecture globale du système.

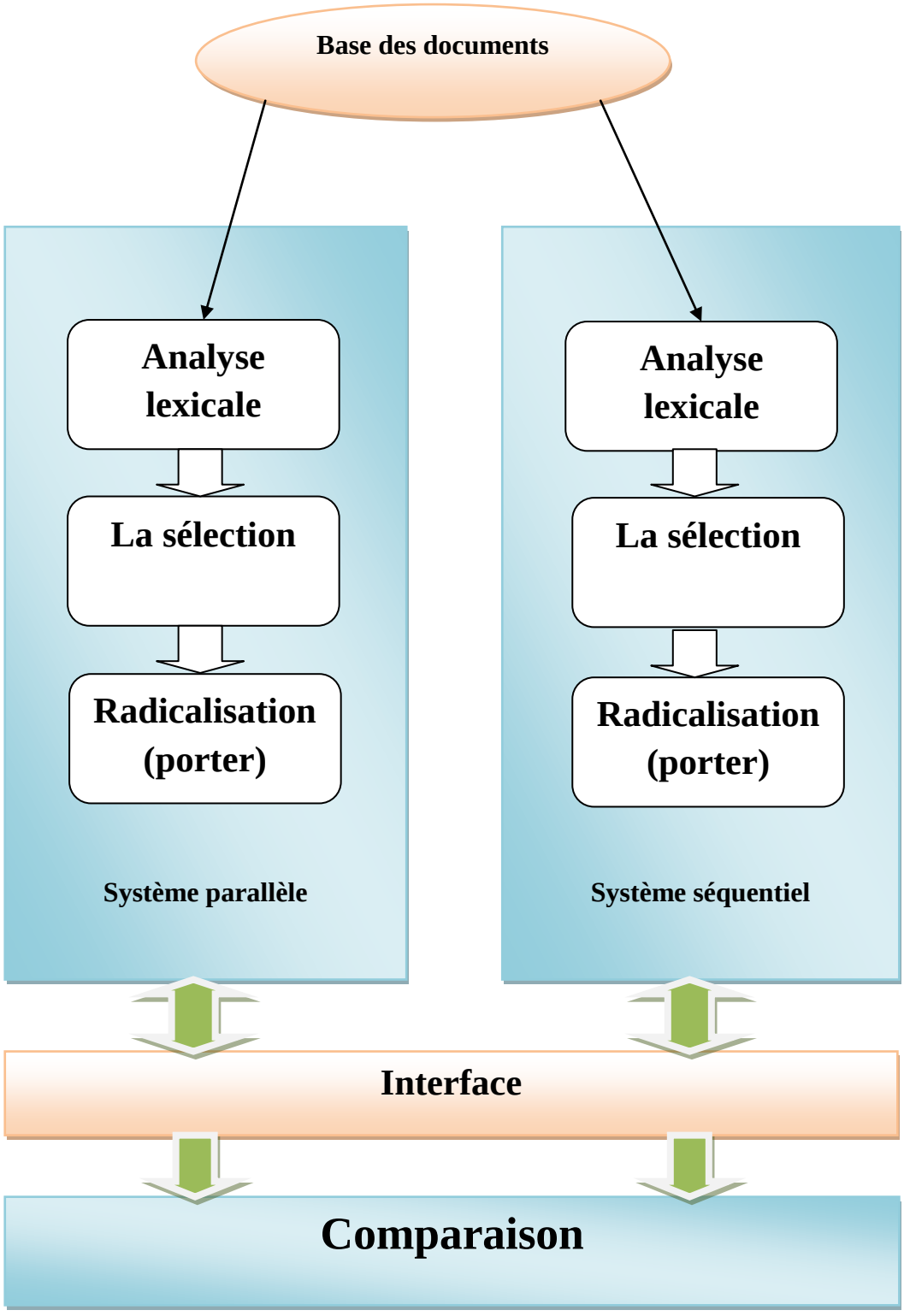


Figure 3.1 *: Architecture globale du système*

3. Principe de fonctionnement du système séquentiel

Le système accède à une base de documents qui contient des documents en anglais, il récupère le texte d'un seul document à la fois et effectue une analyse lexicale, le résultat de cette étape est une liste des termes.

Ensuite, dans l'étape de la sélection ; le système prend chaque terme de la liste obtenu précédemment et le compare avec tous les mots du *stoplist* (anti-dictionnaire), si le mot existe dans le *stoplist* alors le système le supprime sinon il le garde.

Enfin l'étape de la radicalisation: le système prend chaque terme à la fois à partir de la liste qui contient seulement les termes qui ne sont pas vides de sens et applique sur lui l'algorithme de *Porter* qui a pour objective l'extraction du radical. Lors de l'application de cet algorithme on va exécuter les cinq étapes l'une après l'autre séquentiellement (***la figure 3.2)***.Le même principe est appliqué aux autres documents. Puis le système affiche les termes et leurs radicaux et le temps d'exécution du système.

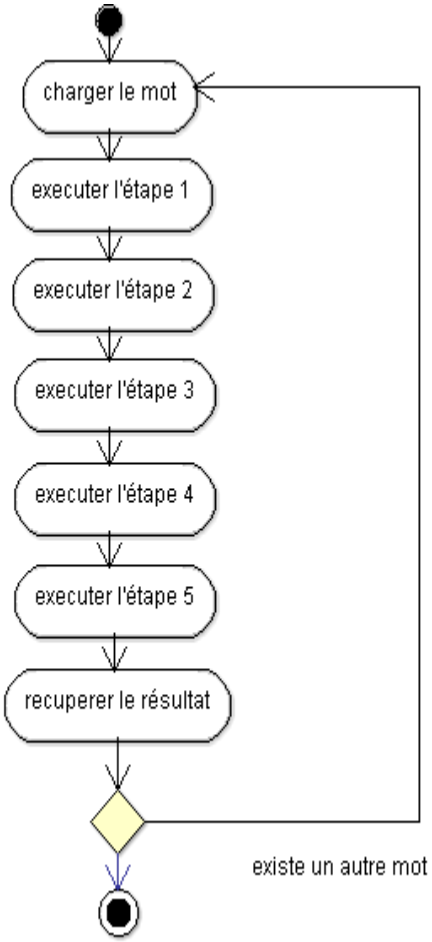


Figure 3.2 : *Diagramme d'activité du système séquentiel*

4. Principe de fonctionnement du système parallèle

L'objectif de ce système est l'extraction des radicaux dans un temps inférieur à celui du système séquentiel. Par conséquence nous avons intégré le parallélisme au niveau de l'indexation dans l'étape de radicalisation (l'algorithme de *Porter*). Pour ce faire, nous avons appliqué quelques principes de réseau de tri *« transposition pair impair »* (vu dans le deuxième chapitre) car ce réseau travaille sur n'importe quelle liste sans contrainte.

Donc à la place d'exécuter une seule étape à la fois (séquentiel) nous avons construit un système parallèle en s'appuyant sur le modèle de programmation parallèles a mémoire partagée (ce modèle est détaillé dans le deuxième chapitre),une architecture SIMD car nous avons les mêmes instructions et des données différentes, une machine parallèle quadruple cœurs pour une exécution parallèle, de cette manière on peut lancer un nombre maximum de threads et chaque thread exécute une seule étape avec un réseau de communication entre les threads, ***La figure 3.3*** représente le Diagramme d'activité de l'exécution parallèle dans le cas de cinq threads (T_1,T_2,T_3,T_4,T_5), chaque thread dispose au début une seule étape respectivement (étape1 , étape2, étape3, étape4 ,étape5), Au début le système charge un mot et l'envoi à tous les threads qui vont s'exécuter en même temps. S'il n'y a aucun changement dans les threads alors le système retourne le résultat. Par contre S'il y a un changement dans un thread T_i, alors ce dernier communique (envoi un message) avec les autres threads (T_{i+1}, Ti_{+2},…..) pour que ces derniers prennent en considération le changement.

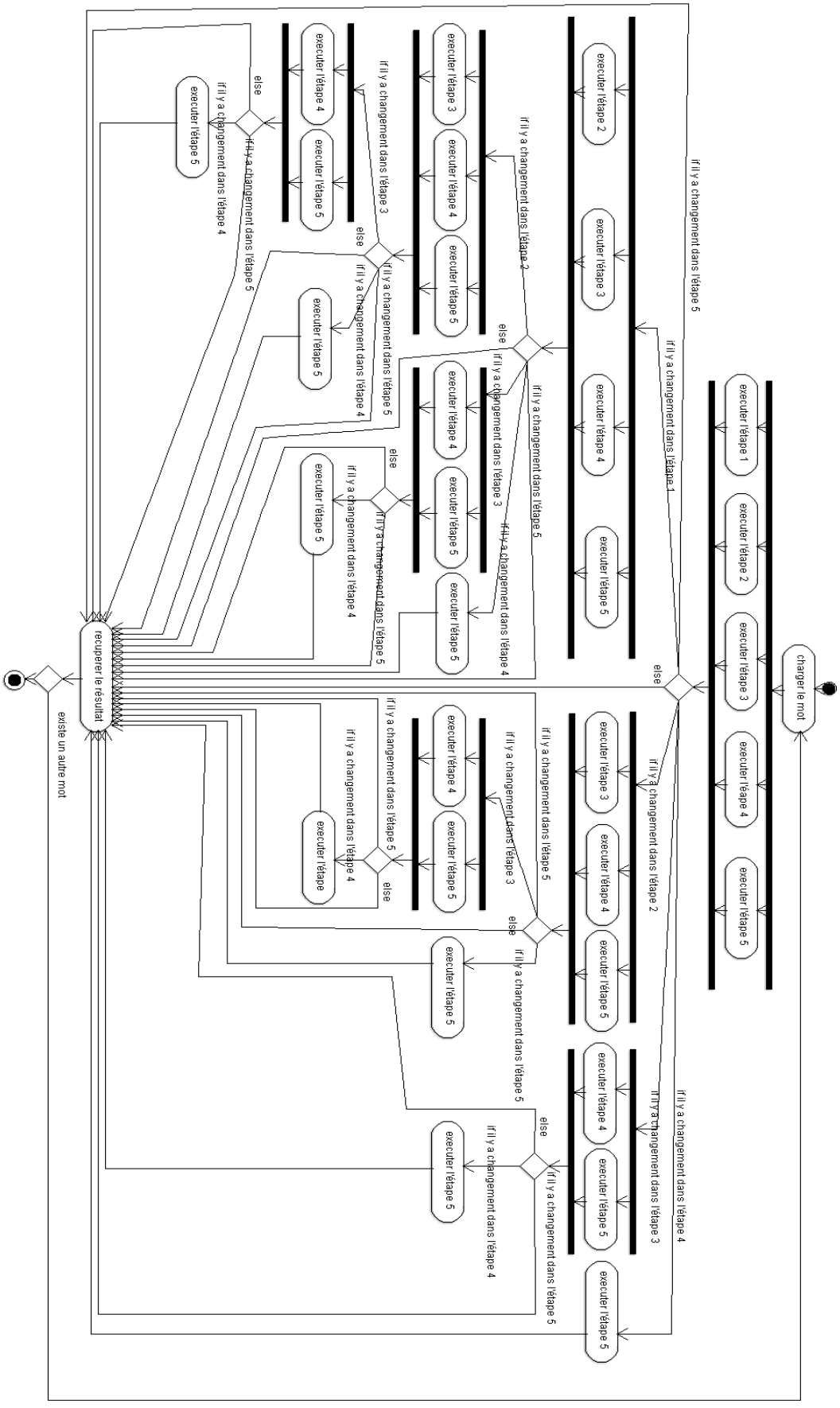


Figure 3.3 : *Diagramme d'activité du système parallèle*

5. Conception du système

Nous avons construit deux systèmes de recherche d'information ; un séquentiel et l'autre parallèle. Les deux systèmes accèdent à la même base de documents et sont gérés par une seule application. Ces systèmes vont nous permettre d'extraire des radicaux de la base de documents qui contient des textes en anglais et ainsi comparer les résultats des deux systèmes on fonction du temps d'exécution.

Les cas d'utilisation (***figure 3.4***) permettent de structurer les besoins des utilisateurs et les objectifs correspondants du système.

Notre système comporte un acteur principal : L'utilisateur

Le système va effectuer l'analyse lexicale, la sélection, puis l'extraction des radicaux et enfin il nous affiche les mots et leurs racines ainsi que le temps d'exécution.

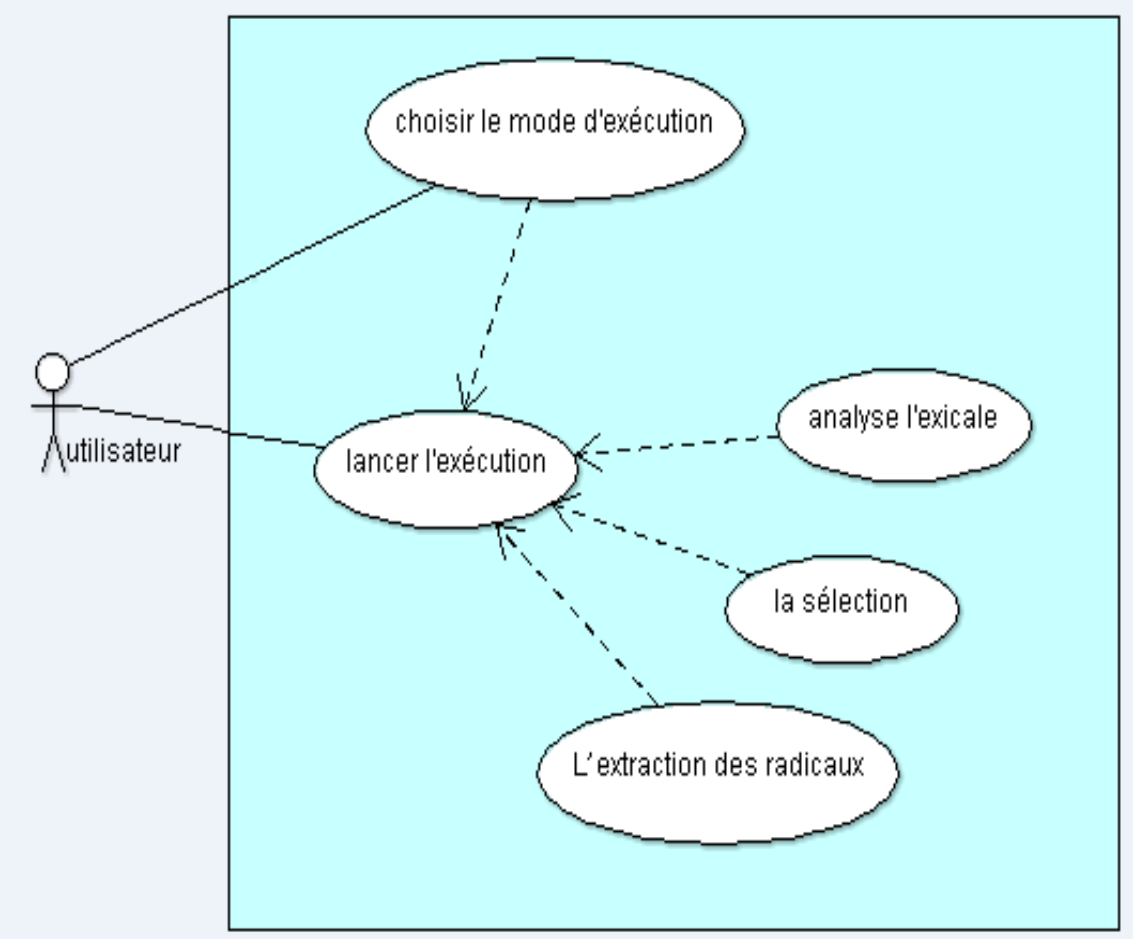


Figure 3.4*: Diagramme de cas d'utilisation*

La figure 3.5 représente un exemple du processus d'indexation.

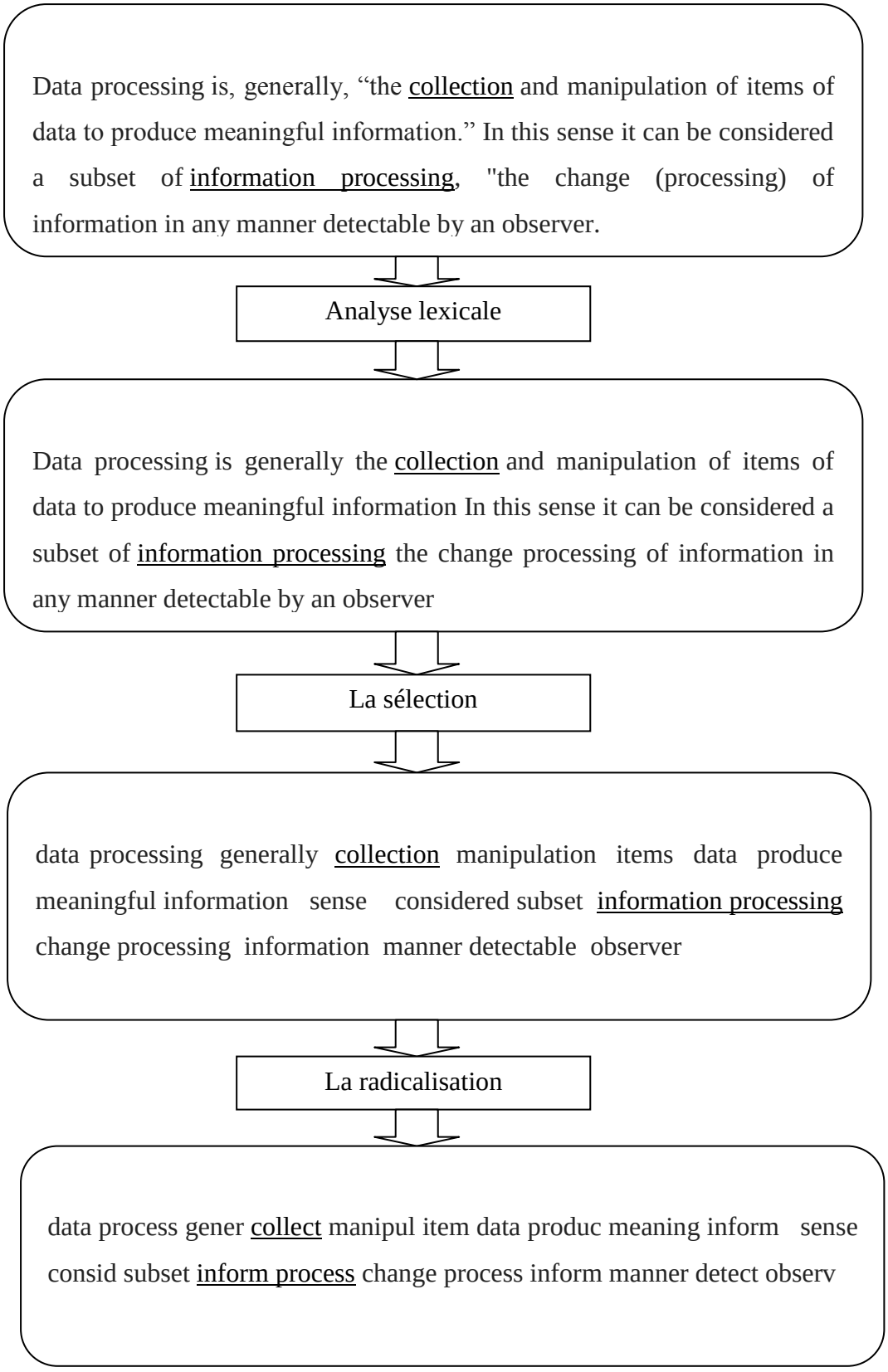


Figure 3.5*: Exemple d’exécution du system d’indexation*

6. Conclusion

Dans ce chapitre nous avons passé en revue le principe de fonctionnement du système séquentiel ainsi que du système parallèle puis la conception du système et un exemple d'exécution du system d'indexation.

CHAPITRE IV :

IMPLEMENTATION DU SYSTEME

CHAPITRE IV : IMPLEMENTATION DU SYSTEME

1. Introduction

Dans ce chapitre nous allons présenter l'interface de notre système puis nous allons exécuter quelques exemples sur les deux systèmes séquentiel et parallèle et enfin nous allons faire une comparaison entres les résultats de ces deux systèmes.

2. L'environnement du système

Le système est réalisé avec le langage de programmation ***JAVA*** sous l'environnement de développement ***éclipse*** et ***PhpMyAdmin*** qui contient des documents textuels numériques rédigés dans la langue anglaise ***(la figure 4.1)***.

Le système sera exécuté sur une machine ***multi-cœur*** (i3).

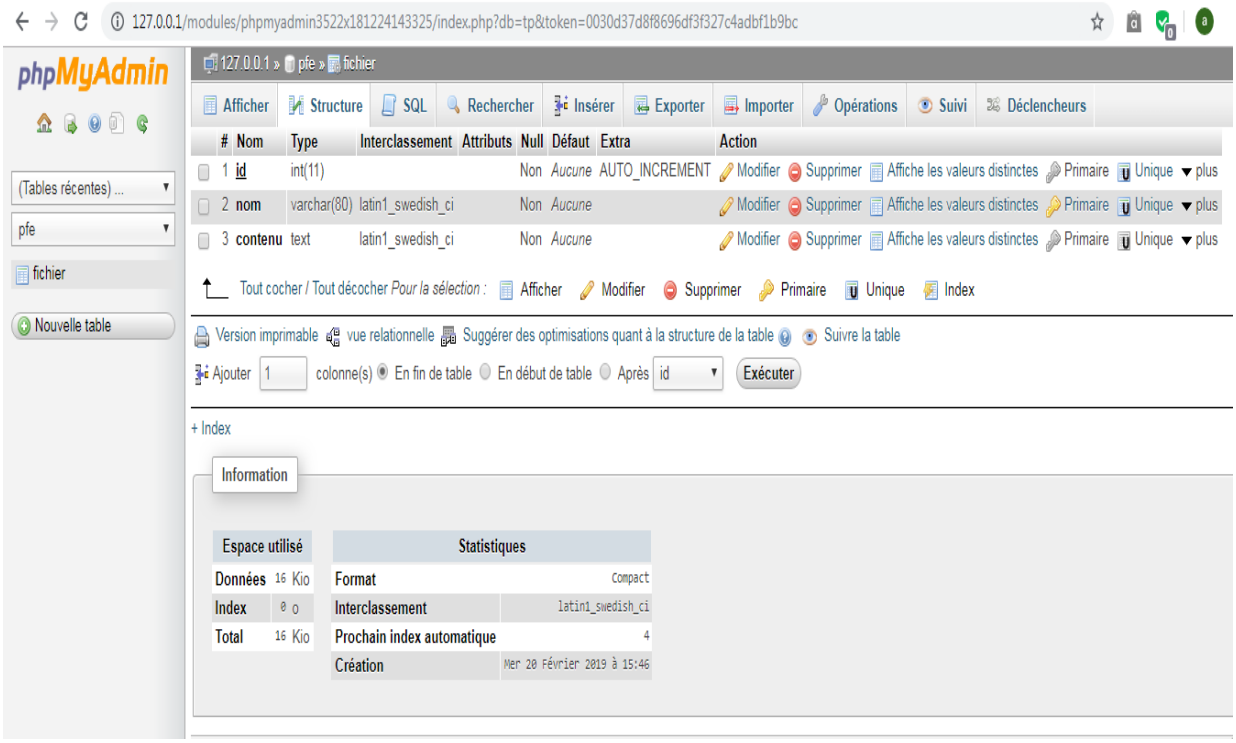


Figure 4.1 : *interface de PhpMyAdmin*

3. Présentation du Système

Nous détaillons ci-dessous l'interface (La ***figure 4.2***) et les fonctionnalités de l'application réalisée.

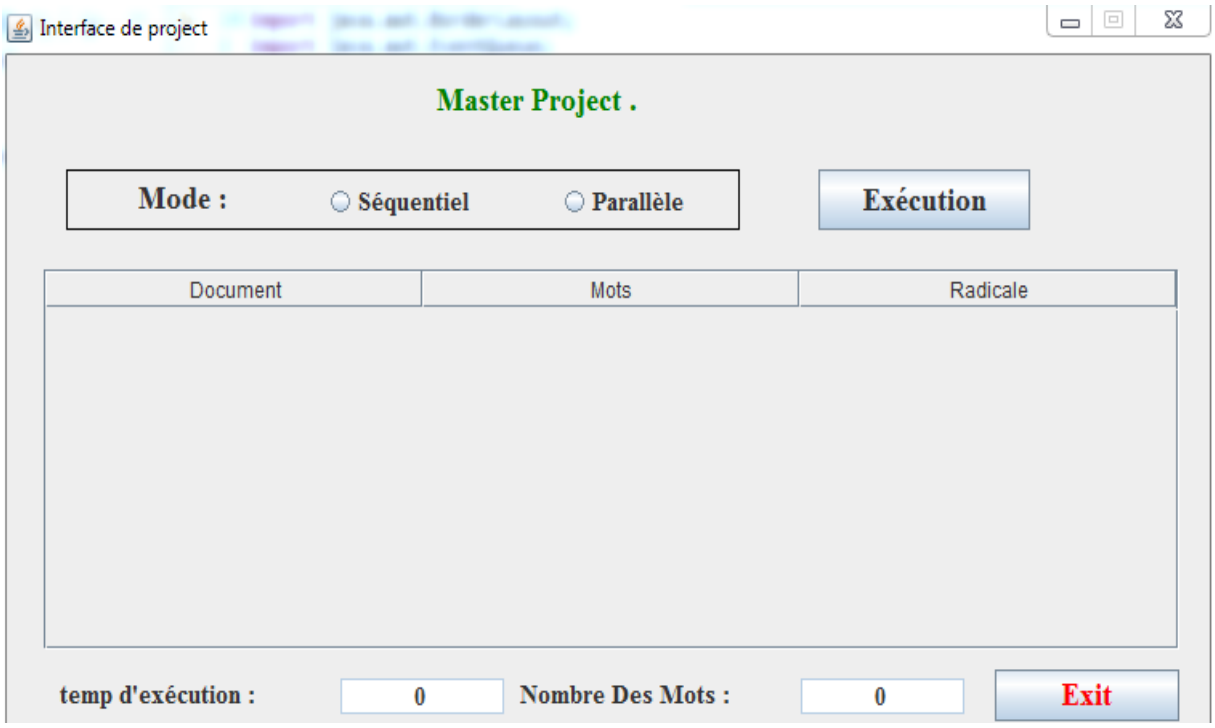


Figure 4.2 : *L'interface de l'application*

La ***figure 4.3*** représente le choix du mode d'exécution de notre système soit séquentiel ou bien parallèle.

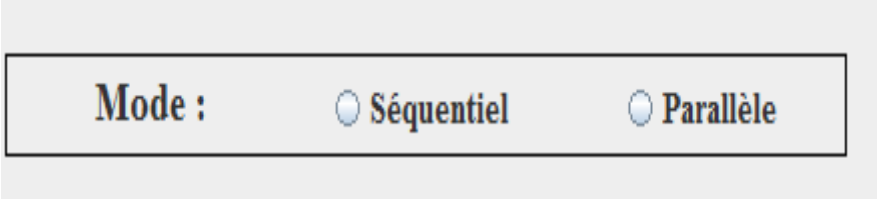


Figure 4.3 : *le mode d'exécution*

Pour que la comparaison en termes de temps d'exécution puisse se faire correctement entre les deux systèmes parallèle et séquentiel, on a exécuté à chaque fois le même exemple sur les deux systèmes dans les mêmes conditions (même base de documents et même machine), nous avons fait un ensemble de tests sur des nombres de mots croissants en fonction du temps.

Les figures ci-dessous montrent les résultats d'exécution pour les deux systèmes pour l'étape d'indexation. Chaque système affiche les termes et leurs racines, il affiche aussi le temps d'exécution écoulé en milliseconde.

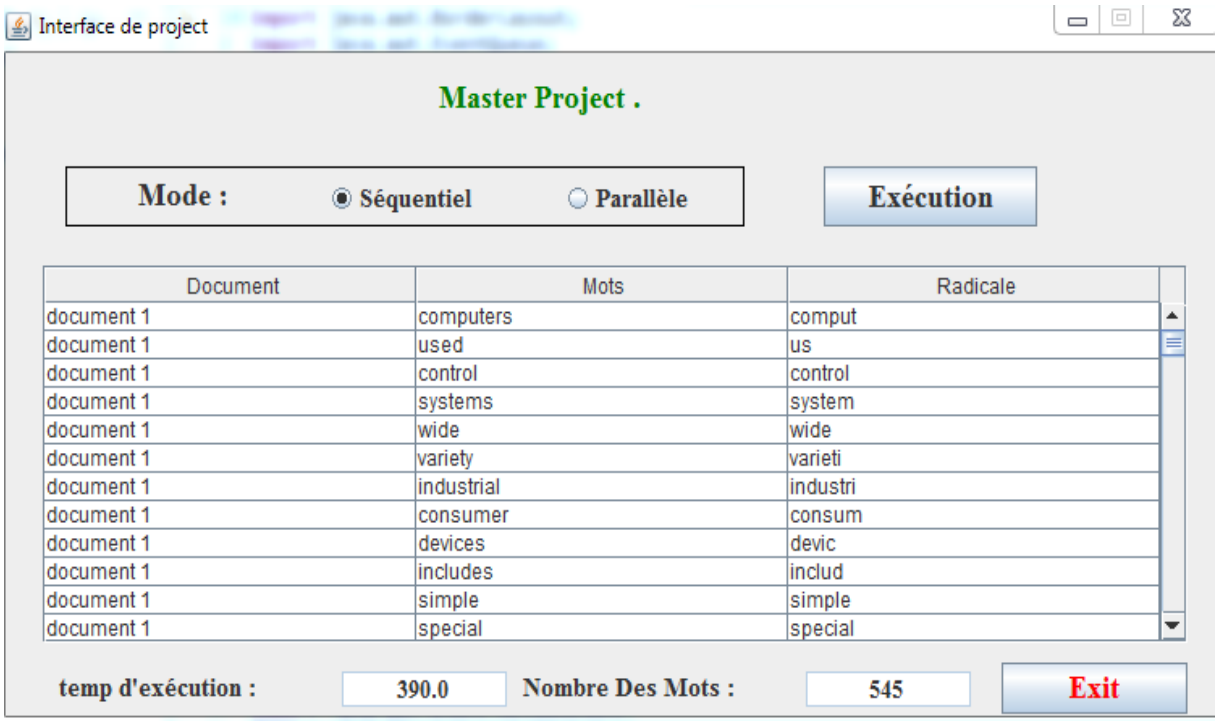


Figure 4.4 : *l'exécution séquentielle pour un nombre des mots égale à 545*

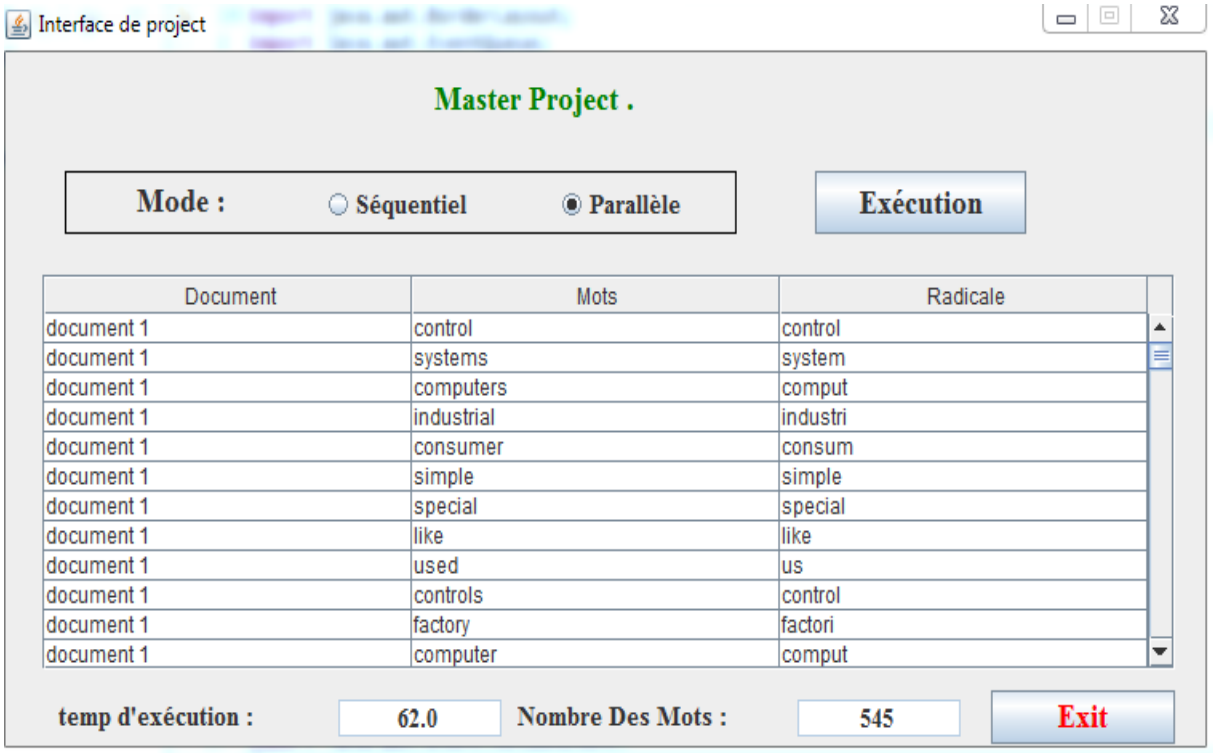


Figure 4.5 : *l'exécution parallèle pour un nombre des mots égale à 545*

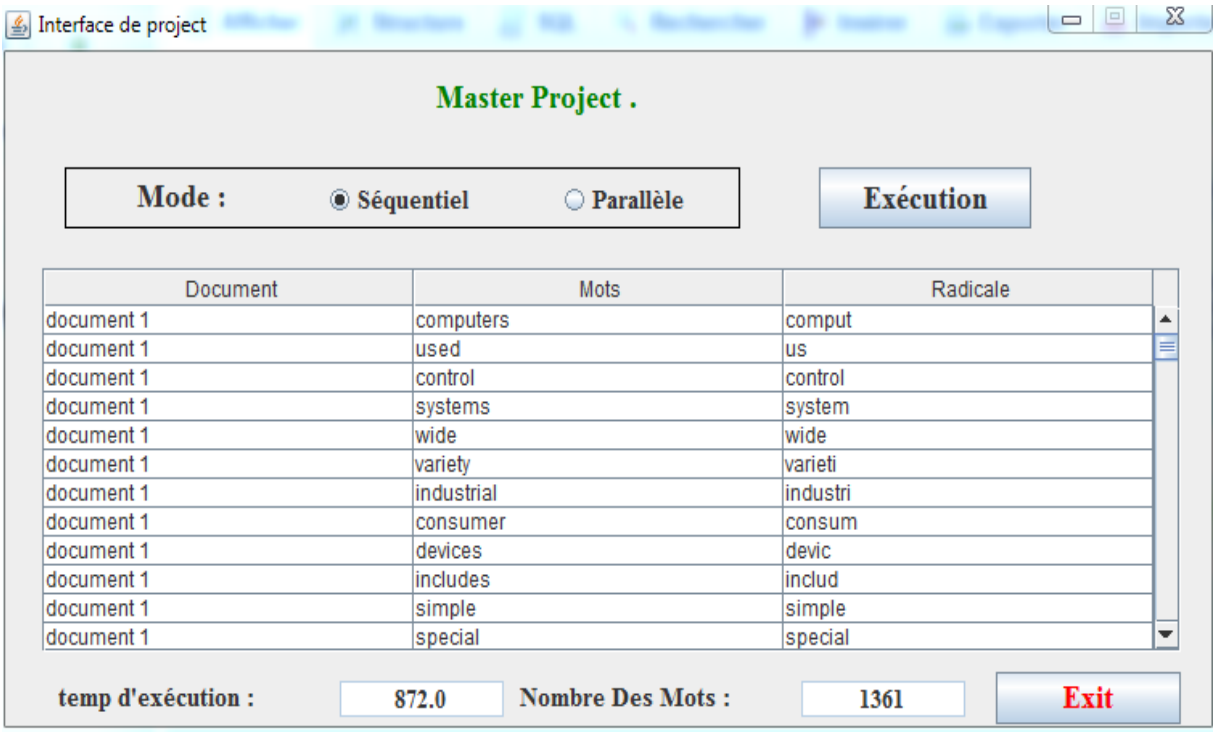


Figure 4.6 : *l'exécution séquentielle pour un nombre des mots égale à 1361*

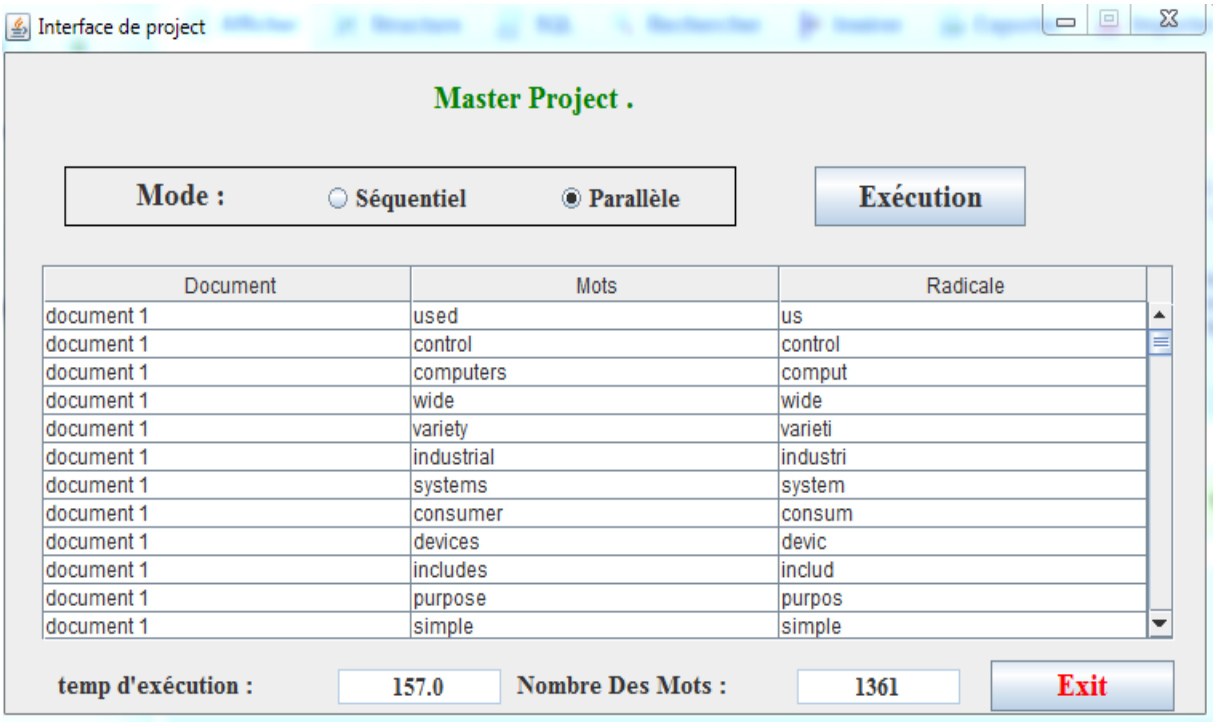


Figure 4.7 : *l'exécution parallèle pour un nombre des mots égale à 1361*

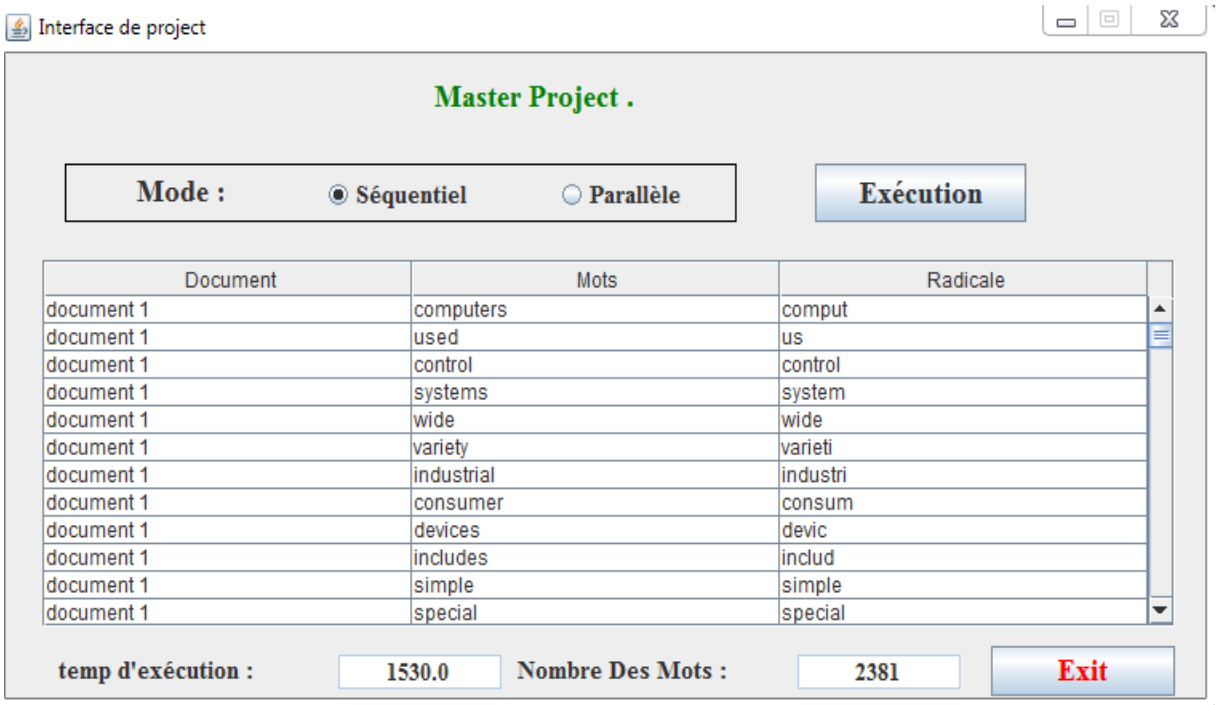


Figure 4.8 : *l'exécution séquentielle pour un nombre des mots égale à 2381*

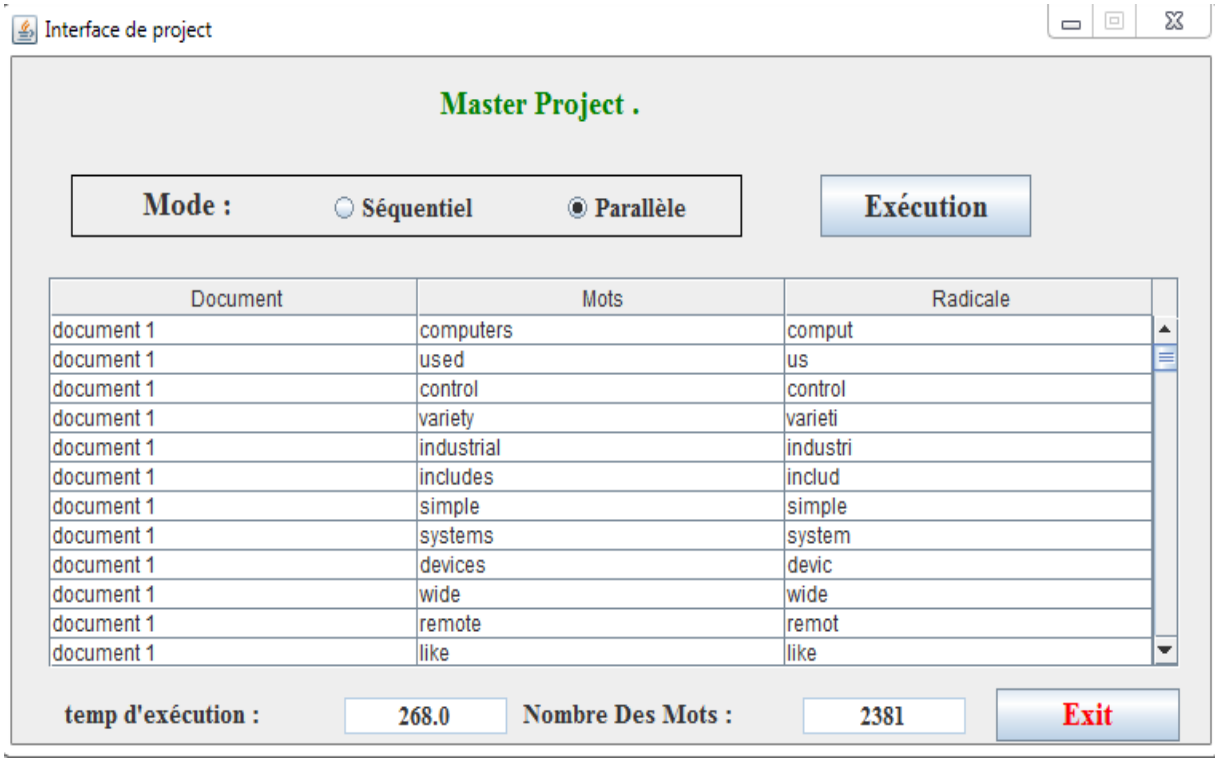


Figure 4.9 : *l'exécution parallèle pour un nombre des mots égale à 2381*

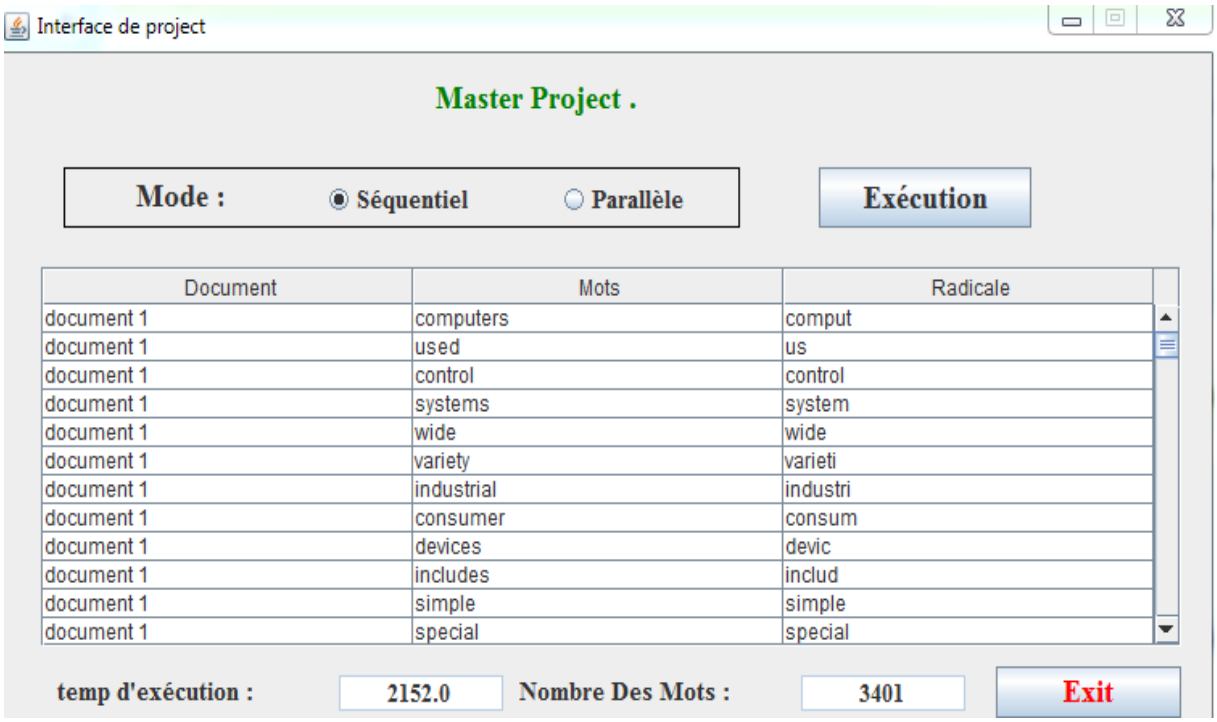


Figure 4.10 : *l'exécution séquentielle pour un nombre des mots égale à 3401*

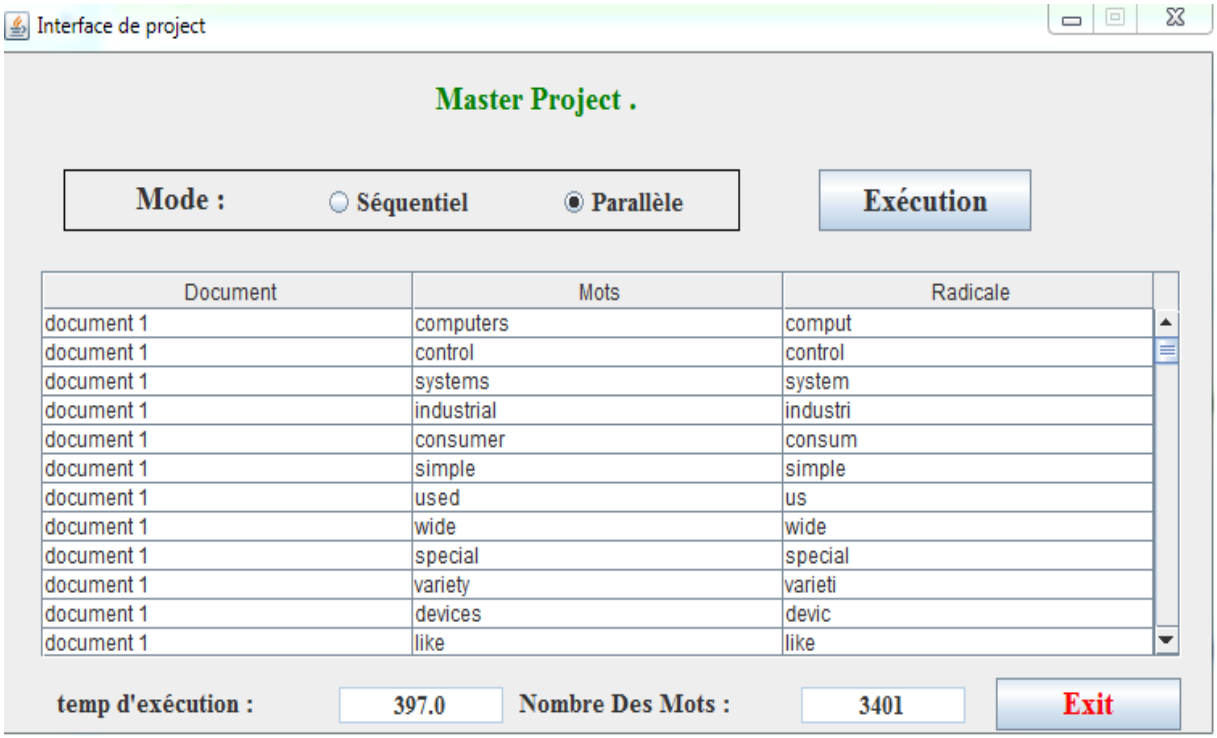


Figure 4.11 : *l'exécution parallèle pour un nombre des mots égale à 3401*

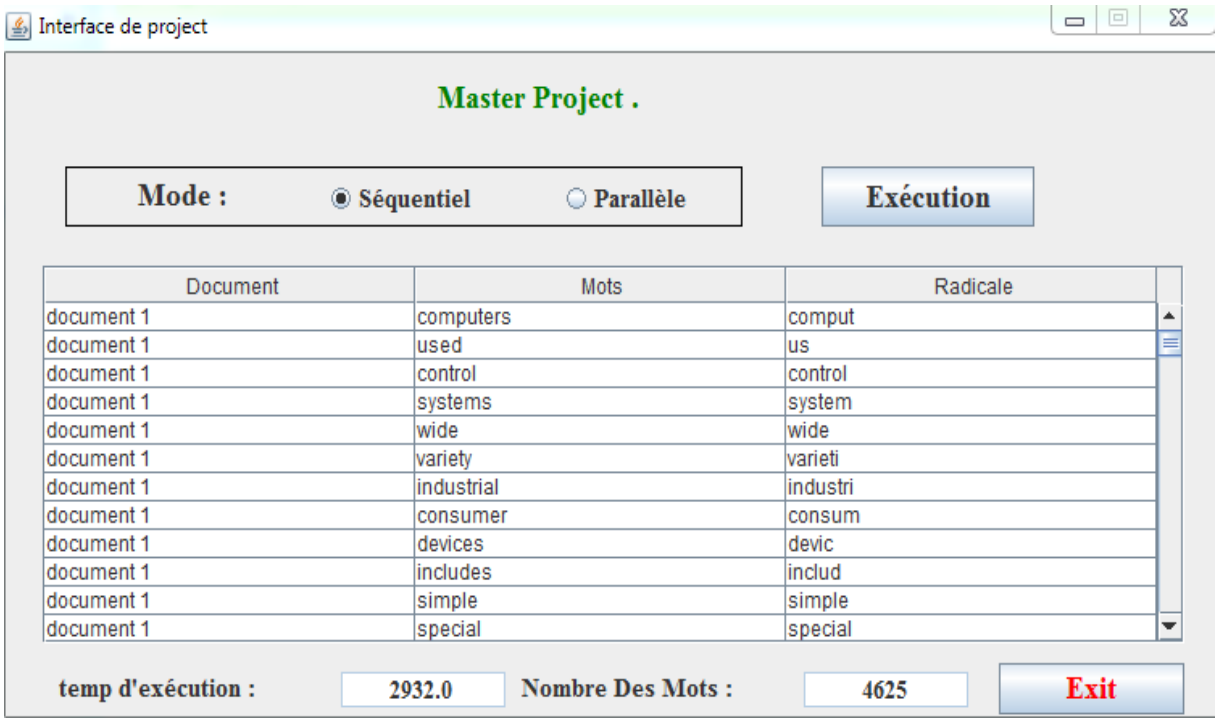


Figure 4.12 : *l'exécution séquentielle pour un nombre des mots égale à 4625*

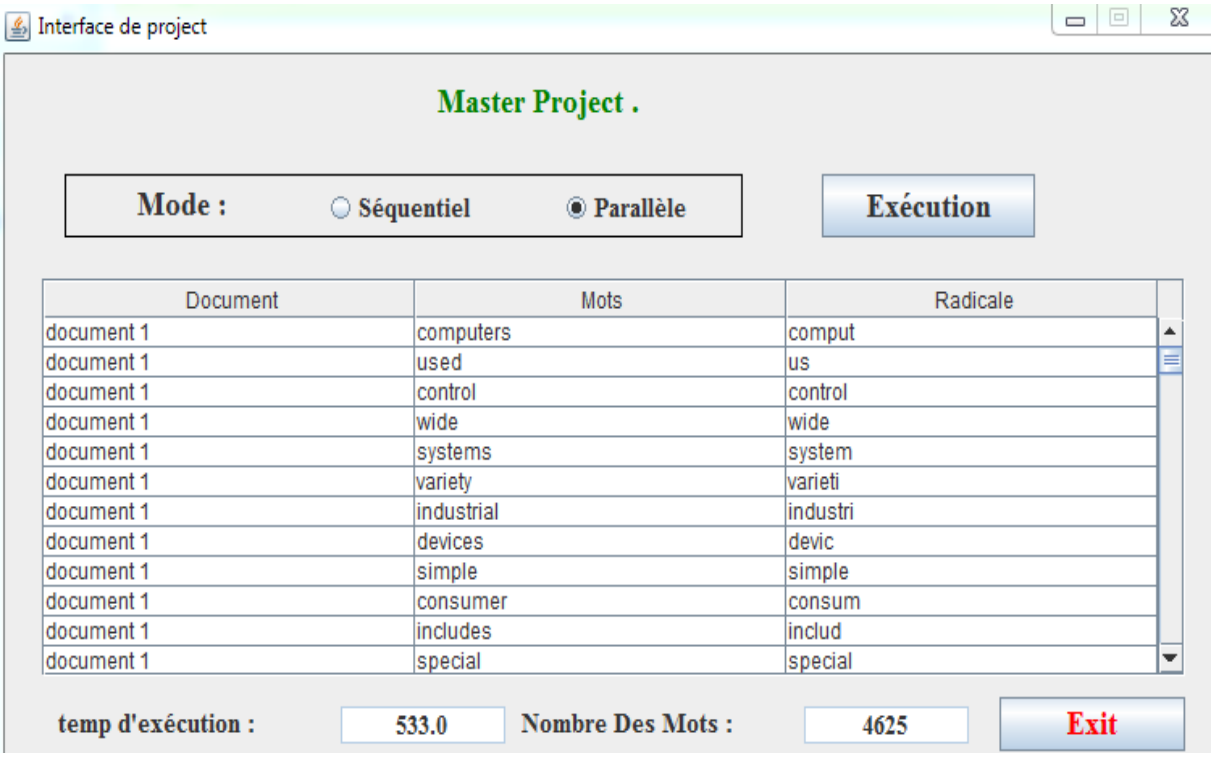


Figure 4.13 : *l'exécution parallèle pour un nombre des mots égale à 4625*

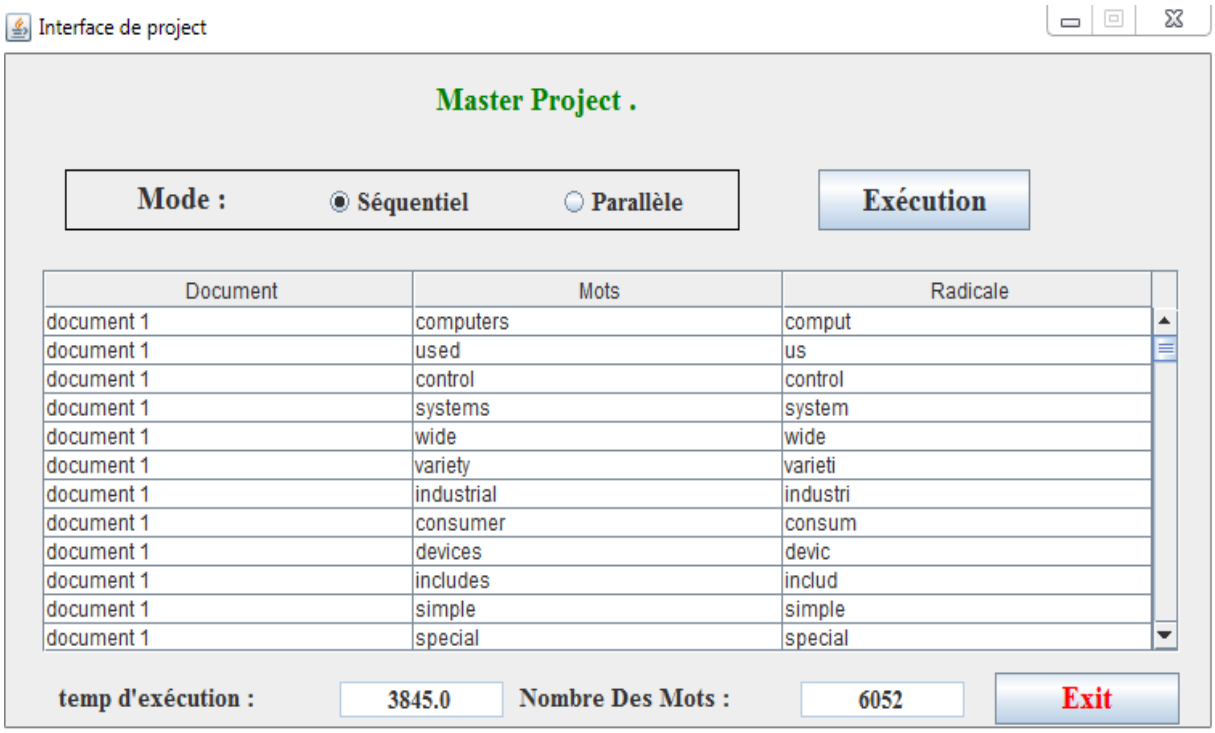


Figure 4.14 : *l'exécution séquentielle pour un nombre des mots égale à 6052*

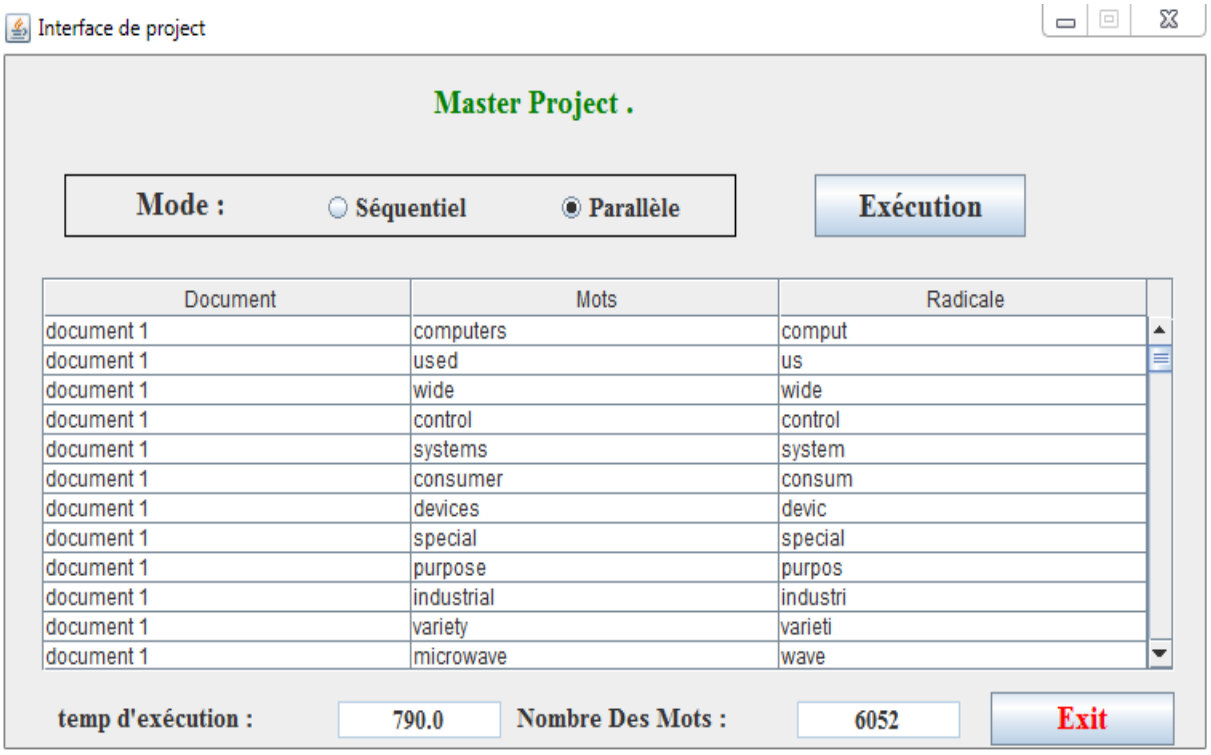


Figure 4.15 : *l'exécution parallèle pour un nombre des mots égale à 6052*

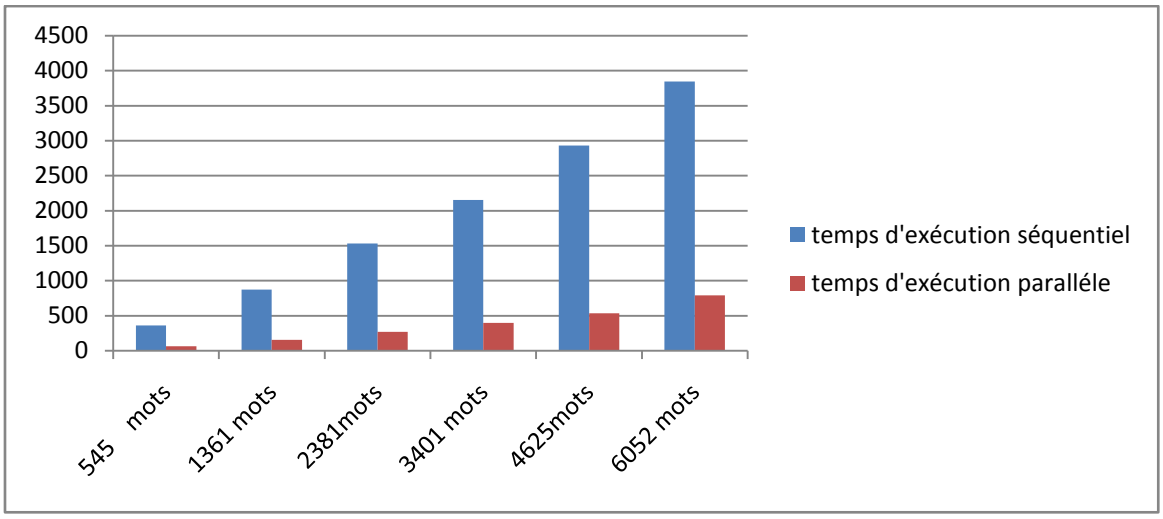
4. Comparaison

On présente les résultats de nos tests obtenus en millisecondes dans le tableau ci-dessous (pour les figures : 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15):

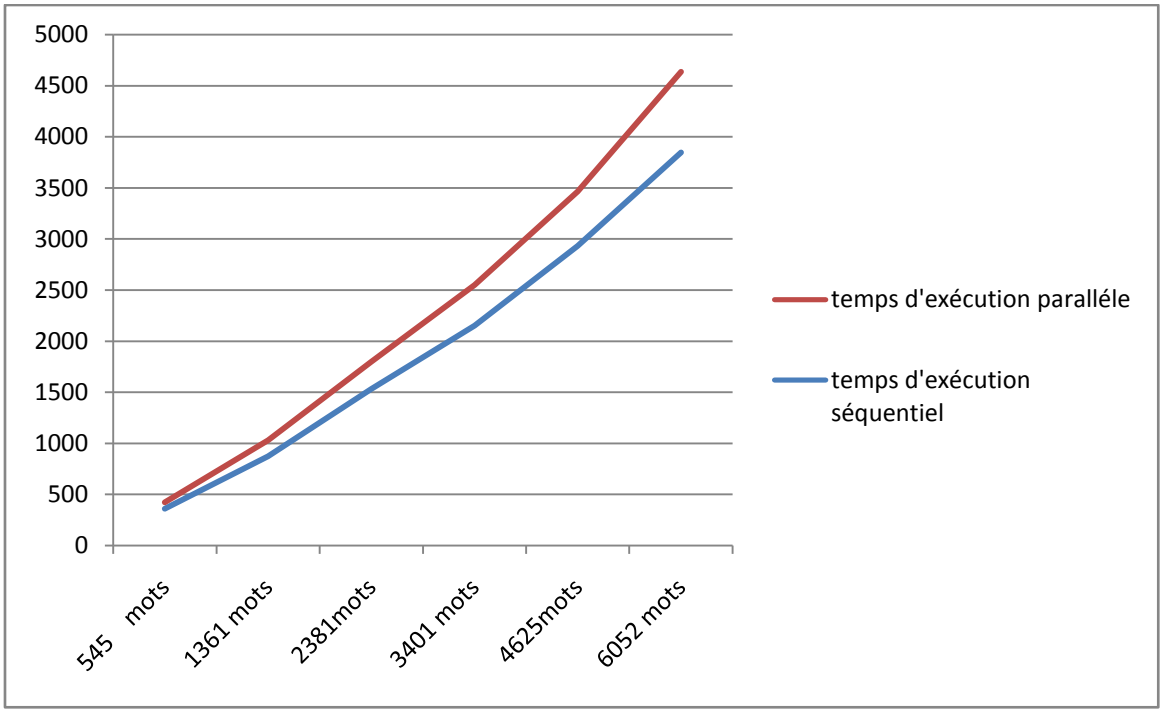
Nombre de mots	545	1361	2381	3401	4625	6052
Système séquentiel	390	872	1530	2152	2932	3845
Système parallèle	62	157	268	397	533	790
L'accélération RI=Ts/Tp	6.29	5.55	5.70	5.42	5.50	4.86

Tableau 4.1: comparaison entre les deux systèmes en fonction du temps.

Ces résultats sont clarifiés et représentés dans les graphes ci-dessous (graphe 1 et graphe 2).



Graphe 01 : Diagramme en barres des temps d'exécution séquentiel et parallèle en millisecondes.



Graphe 02 : Courbes des temps d'exécution séquentiel et parallèle en millisecondes.

On remarque que quelque soit le nombre de mots, le système parallèle est toujours plus efficace en termes de temps d'exécution que le système séquentiel.

On remarque aussi une différence importante entre les deux systèmes en termes d'accélération qui varie entre 4,86 et 6,29.

5. Conclusion

On se basant sur les résultats obtenus ; le système parallèle est non seulement efficace en termes de qualité des résultats obtenus mais aussi beaucoup moins couteux en matière de temps de réponse comparé au système séquentiel.

Conclusion générale

Nous avons essentiellement présenté d'une manière générale dans ce travail les notions de base de la recherche d'information. Nous avons vu aussi les différents modèles de la RI en soulignant les avantages et les limites de chacun d'eux.

Etant donné que ce travail est basé sur le processus d'extraction des radicaux des termes qui est une étape de l'indexation, on a présenté ces différentes étapes qui les composent cette dernière.

Nous avons choisi l'algorithme de Porter (pour l'extraction du radical), et se basant sur les principes du parallélisme on a construit un système d'indexation (n'incluant pas la partie pondération) afin d'améliorer les performances en termes de temps.
Notre système réalisé dans ce travail permet l'exécution de plusieurs tâches en parallèle ce qui nous a fournis un gain de temps remarquable en comparant avec la version séquentielle du même système et dans les mêmes conditions.

Perspectives

Il existe plusieurs études et recherches qui montre que la plupart des gens utilise un seul mot pour la recherche d'information et ça peut conduire a des résultats erronés L'une des éventuelles améliorations qu'on peut apporter au système de recherche d'information parallèle est l'introduction de la sémantique des termes et l'utilisation des notions des machine Learning afin de minimiser le choix de réponse et augmenter la possibilité d'obtenir les résultats souhaités.

Références bibliographiques

1. Mooers, C.N., Application of Random Codes to the Gathering of Statistical Information, MIT Master's Thesis, 1948.
2. Salton, G., McGill, M. Introduction to Modern Information Retrieval. McGraw-Hill Int. Book Co, 1984.
3. Hernandez N. "Ontologie de domaine pour la modélisation du contexte en recherche d'information", thèse de doctorat en informatique, Université Paul Sabatier. (2006)
4. Boubekeur F. "Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets", thèse de doctorat en informatique, Université Paul Sabatier. 2008
5. Fuhr, N. Information Retrieval - From Information Access to Contextual Retrieval. In M. Eibl, C. Wolf, and C. Womser-Hacker, editors, Designing Information Systems. Festschrift für Jürgen Krause, pp. 47-57. UVK Verlagsgesellschaft, 2005.
6. Salton, G., E.A. Fox, H. Wu. Extended Boolean information retrieval system. CACM 26(11), pp. 1022-1036, 1983.
7. P. Ingwersen. "Polyrepresentation of information needs and semantic entities: elements of a cognitive theory for information retrieval interaction". In Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval., pages 101-110, 1994.
8. Cleverdon, C. Progress in documentation. Evaluation of information retrieval systems. Journal of Documentation, 26, pp. 55–67, 1970.
9. Harter SP. 1992. Psychological relevance and information science. Journal of the American Society for Information Science (1986-1998), 43 : 602.
10. Borlund P etIngwersen P. Measures of relative relevance and ranked half-life: performance indicators for interactive IR. Dans : Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 1998. ACM, p. 324-331.
11. Saracevic T. Relevance reconsidered. Dans : Information science: Integration in perspectives In Proceedings of the Second Conference on Conceptions of Library and Information Science, 1996. p. 201- 218

12. C. Tambellini. “Un système de recherche d’information adapté aux données incertaines: adaptation du modèle de langue”. Thèse de doctorat en informatique, Université de Nice-Sophia Antipolis-UFR sciences, 2007.
13. Ren, F., Fan, L., Nie, J-Y. SAAK Approach: How to Acquire Knowledge in an Actual Application System. International Conference on Artificial Intelligence and Soft Computing, Honolulu, pp.136-140, 1999.
14. Jacquemin, C., Daille, B., Royanté, J., and Polanco, X. In vitro evaluation of a program for machine-aided indexing. Inf. Process. Manage. 38, 6, pp. 765-792. 2002.
15. M. Porter, « An algorithm for suffix stripping,» Program: electronic library and information, pp. p 211-218, 2006.
16. Kabil, B. (2013) : Un Nouvel Algorithme de Stemmatisation pour l’Indexation Automatique de Documents non-structurés : StemmerSAID.
17. Paternostre, M., Francq, P., Lamoral, J., Wartel, D. , et Saerens, M. (2002) : Carry, un algorithme de désuffixation pour le français.
18. J. B. Lovins, «Development of a stemming algorithm,» Journal of Mechanical Translation and Computational Linguistics, pp. pp. 22-31, 1968.
19. Baeza-Yates R etRibeiro-Neto B. 1999. Modern information retrieval. ACM press New York.
20. Dominich, S. Mathematical Foundations of Information Retrieval. Kluwer Academic Publishers, Dordrecht, Boston, London, 2001.
21. Salton G, Fox EA et Wu H. 1983. Extended Boolean information retrieval. Communications of the ACM, 26 : 1022-1036.
22. Champclaux, Y. (2009) Un modèle de recherche d’information basé sur les graphes et les similarités structurelles pour l’amélioration du processus de recherche d’information (thèse)
23. Salton, G. The smart Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall, 1971.
24. Van Rijsbergen, C. J. Information retrieval. London: Butterworth, 1979.
25. .Robertson, S. E. The Probability Ranking Principle in IR. Journal of Documentation, 33(4), pp. 294-304, 1977.
26. Carpineto, C., Romano, G., «A Survey of Automatic Query Expansion in Information Retrieval». ACM Computing Surveys, Vol. 44, No. 1. 2012.

27. M. Flynn. Some Computer Organizations and Their Effectiveness. IEEE Trans. Comput., C-21 :948+, 1972.
28. El-Rewini, Hesham; Abd-El-Barr, Mostafa (2005). *Advanced Computer Architecture and Parallel Processing*. Wiley-Interscience. pp. 77–80.
29. Christoph Lameter, Phd. NUMA becomes more common because memory controllers get close to execution units on microprocessors.9 aout 2013.
30. Per stenströms truman joe and anoop gupta ,Comparative Performance Evaluation of Cache Coherent NUMA and COMA Architectures ,Computer Systems Laboratory Stanford University CA 94305.
31. José Duato, sudhakar yalamanchili, Lionel Ni, in Interconnection Networks,Morgan Kaufmann, 2003 - 600 pages.