République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Batna
Faculté des Sciences

Thèse

En vue de l'obtention du diplôme de

Doctorat en Sciences en Informatique

Architecture Parallèle pour l'accélération de la recherche dans les routeurs IP

Présentée Par

Hichem HOUASSI

Soutenue le 06/06/2012

Devant le jury composé de :

Président	*Pr. BOUGUECHAL Nouredine*	*Professeur UHL Batna*
Directeur de thèse	*Pr. BILAMI Azeddine*	*Professeur UHL Batna*
Examinateurs	*Pr. BENMOHAMMED Mohamed*	*Professeur UMC Constantine*
	Pr. CHAOUI Allaoua	*Professeur UMC Constantine*
	Dr. BELATTAR Brahim	*M.C.A UHL Batna*

REMERCIEMENTS

Grand merci à Dieu.

Cette thèse représente un travail mené pendant plusieurs années. Il serait très difficilement réalisable sans le soutien et l'accompagnement de l'entourage. C'est pourquoi Je tiens à remercier toutes les personnes qui m'ont aidé durant ces années de thèse.

Je remercie plus particulièrement Monsieur BILAMI Azeddine, Professeur à l'Université Hadj Lakhdar Batna, pour m'avoir encadré. Je le remercie pour sa pédagogie, son soutien et pour toutes les discussions instructives et fructueuses menées durant toute la durée de ce travail de recherche.

Je remercie vivement Monsieur BOUGUECHAL Noureddine, Professeur à l'Université Hadj Lakhdar Batna, pour avoir accepté de présider le jury de cette thèse.

Mes remerciements sont aussi pour Messieurs BENMOHAMMED Mohamed, Professeur à l'Université Mentouri Constantine, CHAOUI Allaoua, Professeur à l'Université Mentouri Constantine, et BELATTAR Brahim Maitre de conférences à l'Université Hadj Lakhdar Batna, pour l'intérêt qu'ils portent à ces travaux, et en acceptant de faire partie du jury de cette thèse.

Je remercie vivement ma famille de m'avoir laissé travailler durant autant d'années sans (trop) poser de questions et de m'avoir fait confiance.

Enfin, je souhaite remercier ma femme, pour son soutien et sa compréhension, qui m'ont permis de venir à bout de mes objectifs.

Résumé

Le réseau Internet est composé de nœuds et de routeurs reliés ensemble par des médias de transmission (liens). Les paquets de données transitent sur les liens, d'un routeur à un autre, vers leurs destinations finales. Chaque routeur exécute une décision d'expédition sur les paquets entrants pour déterminer le prochain routeur du paquet.

Le routeur Internet est un dispositif permettant de choisir le chemin que les paquets de données vont emprunter pour arriver à leurs destinations. Pour ce faire, le routeur effectue une recherche dans la table de routage en utilisant l'adresse de destination d'un paquet IP comme une clé ; cette tâche du routeur est appelée décision de routage. La croissance du nombre d'utilisateurs d'Internet à comme conséquence l'augmentation des tailles des tables de routage des routeurs et la complexité de l'opération d'expédition des paquets de données.

La décision de routage consiste à chercher la meilleure route pour un paquet à partir de son adresse IP destination. Afin d'augmenter l'efficacité du routage dans l'Internet, plusieurs adresses destination sont agrégées en une seule entrée dans une table de routage (préfixe). Depuis l'avènement du CIDR (*Classless Inter-Domain Routing*), les préfixes dans la table de routage ont des longueurs variables, alors plusieurs préfixes peuvent être associés à une même adresse IP de destination, le routeur doit choisir le plus long préfixe correspondant (PLP) à l'adresse IP. Etant donné que les préfixes dans la table de routage ont des longueurs différentes, alors, nous ne pouvons pas trouver la correspondance en utilisant un algorithme de recherche avec une correspondance exacte. Par conséquent, les performances du routeur dépendent fortement de l'efficacité de l'opération de recherche du plus long préfixe dans la table de routage.

Différentes approches logicielles et matérielles ont été proposées et déployées pour améliorer les performances des routeurs, et dont la métrique de performance a été principalement le débit ou le nombre de paquets expédiés par seconde. Dans cette thèse nous avons proposé trois mécanismes de recherche du plus long préfixe correspondant à une adresse IP dans une table de routage, les mécanismes que nous avons proposés dans cette thèse contribuent à améliorer la performance de l'acheminement de paquets dans les routeurs IP.

Mots clés:

Routeur IP, Table de routage, plus long préfixe, architecture parallèle, description VHDL, algorithme parallèle, expansion des préfixes, table de routage cache, arbre binaire, arbre binaire à contenu dynamique.

Abstract

The Internet network is composed of nodes and routers connected by transmission media (links). The data packets pass on links via routers until reaching their final destinations. Every router performs a packet forwarding decision on the incoming packet in order to determine the next-hop router. To do this, the router searches the routing table using the destination IP address of an IP packet as a key; this task is called the routing decision. The growing number of Internet users lead to the growing of the routing table sizes and the increasing of the complexity of the forwarding data packet operation.

Considering some criterion, the routing decision chooses the best route for a packet for its destination IP address. To increase the efficiency of routing in the Internet, several destination addresses are aggregated into a single entry in a routing table (prefix). Since the advent of CIDR (Classless Inter-Domain Routing), prefixes in the routing table became with variable lengths, so that multiple prefixes may be associated with the same destination IP address, the router must choose the longest matching prefix (LMP) to the IP address. Since the prefixes in the routing table have different lengths, we cannot find a matching algorithm using an exact match search. Therefore, router performance depends heavily on the effectiveness of the longest prefix search operation in the routing table.

Different hardware and software approaches have been proposed and deployed to improve the performance of routers, primarily evaluated in terms of the number of packets forwarded per second. In this thesis we propose three mechanisms of the IP address search operation in a routing table. The proposed mechanisms contribute to improve the performance of packet forwarding in IP routers.

Key words:
IP Router, Routing table, longest prefix, parallel architecture, VHDL description, parallel algorithm, Prefix expansion, Cache routing table, IP address lookup, binary trie, dynamic content binary trie.

Table des matières

Chapitre 2 : Travaux dans le domaine

Chapitre 3 : Parallélisme et architectures parallèles

Chapitre 4 : L'architecture parallèle « PARIR »

Chapitre 5 : Solutions logicielles proposées

Liste des figures

Liste des Tableaux

Liste des Abréviations

A

ABCD	**A**rbre **B**inaire à **C**ontenu **D**ynamique
ALU	**A**rithmetic and **L**ogical **U**nit
APC	**A**ligned-**P**refix **C**aching
AS	**A**utonomous **S**ystem
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuits

B

BMP	**B**est **M**atching **P**refix
BPT	**B**inary **P**refix **T**ree

C

CAM	**C**ontent-**A**ddressable **M**emory
CAO	**C**onception **A**ssistée par **O**rdinateur
CC-NUMA	**C**ache-**C**oherent **N**on-**U**niform **M**emory **A**ccess
CIDR	**C**lassless **I**nter-**D**omain **R**outing
CM	**C**onnexion **M**achine
CPU	**C**entral **P**rocessing **U**nit

D

DA	**D**estination **A**dress
DPT	**D**isjoint **P**refix **T**ree

E

E-BMP	**E**qual **B**est **M**atching **P**refix

F

FAI	**F**ournisseur d'**A**ccès à **I**nternet
FD	**F**lot de **D**onnées
FE	**F**orwording **E**ngine
FI	**F**lot d'**I**nstructions
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FTP	**F**ile **T**ransfer **P**rotocol

G

G-BMP	**G**reater **B**est **M**atching **P**refix

H

HAC	**H**ost **A**ddress **C**ache
HDL	Hardware Description Language
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol

I

IANA	**I**nternet **A**ssigned **N**umbers **A**uthority
I-BGP	**i**nterior **BGP**
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IGMP	**I**nternet **G**roup **M**anagement Protocol
IP	**I**nternet **P**rotocol
IPv4	**I**nternet **P**rotocol **V**ersion **4**
IPV6	**I**nternet **P**rotocol **V**ersion **6**
IS-IS	**I**ntermediate **S**ystem To **I**ntermediate **S**ystem
ISO	**I**nternational **O**rganization for **S**tandardization

L

L-BMP	**L**ess **B**est matching **P**refix
LC-Trie	**L**evel **C**opressed **T**rie
LPFST	**L**ongest **P**refix **F**irst **S**earch **T**ree
LSA	**L**ink-**S**tate **A**dvertisement

M

MAC	**M**edia **A**ccess **C**ontrol
MBPT	**M** Binary **P**refix **T**ree
ME	**M**inimal **E**xpansion
MISD	**M**ultiple **I**nstruction stream, **S**ingle **D**ata stream
MIMD	**M**ultiple **I**nstructions stream, **M**ultiple Data stream
MP	**M**émoire **P**artagée
MPC	**M**ultizone **P**ipelined **C**ache

N

NART	**N**etwork **A**ddress **R**outing **T**able
NCP	**N**etwork **C**ontrol **P**rotocol
Nnv	**N**ombre de **N**œuds **V**ide
Np	**N**ombre de **p**réfixe
NPd	**N**ombre de **P**réfixes **D**éplacé
NTn	**N**ombre **T**otal de **N**œuds

O

OSPF	**O**pen **S**hortest **P**ath **F**irst

P

PATRICIA	**P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric
PARIR	**P**arallel **A**rchitecture for **R**outing **I**nformation **R**esearch
PC-Trie	**P**ath **C**ompressed **T**rie
PRAM	**P**arallel **R**andom **A**ccess **M**achine
PS	**P**ort de **S**ortie

R

RAM	**R**andom **A**ccess **M**emory
RFC	**R**equest **F**or **C**omments
RIP	**R**outing **I**nformation **P**rotocol
RISC	**R**educed **I**nstruction **S**et **C**omputer
RRC	**R**everse **R**outing **C**ache
RTL	**R**esistor-**T**ransistor **L**ogic

S

SIMD	**S**ingle **I**nstruction stream, **M**ultiple **D**ata stream
CISC	**C**omplex **I**nstruction **S**et **C**omputer
SISD	**S**ingle **I**nstruction stream, **S**ingle **D**ata stream
SMP	**S**ymmetric **M**ulti-**P**rocesseurs
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SPMD	**S**ingle **P**rogram **M**ultiple **D**ata
SRAM	**S**tatic **R**andom **A**ccess **M**emory

T

TCAM	**T**ernary **C**ontent-**A**ddressable **M**emory
TCP	**T**ransmission **C**ontrol **P**rotocol

TTL **T**ime **T**o **L**ive

U

UC **U**nité de **C**ontrôle
UDP **U**ser **D**atagram **P**rotocol
UR **U**nité de **R**echerche)
UT l'**U**nité de **T**raitement

V

VHDL **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage

VLSM **V**ariable-**L**enght **S**ubnet **M**asking
VOIP **V**Oix sur **r**éseau **IP**

W

WPT **W**eigted **P**refix **T**ree

Chapitre 1

Introduction

Le routeur prend l'adresse IP de destination d'un paquet et cherche dans sa table de routage les informations de routage, cette opération s'appelle consultation d'adresse IP (address lookup). Dans ce contexte, notre travail s'oriente vers l'accélération de cette opération. Dans ce chapitre nous présentons le contexte de travail qui est la recherche d'information de routage dans les routeurs Internet et nous présentons les motivations, les objectifs et les contributions de cette thèse.

1.1. Introduction

Un réseau peut être vu comme un ensemble de ressources mises en place pour offrir un ensemble de services. Dans l'Internet, la communication entre machines est effectuée en utilisant les paquets d'informations. Une fois que les machines émettent leurs paquets dans le réseau, ce sont les routeurs qui retransmettent ces paquets sur les liaisons des réseaux pour les acheminer vers leurs destinations finales. La croissance du nombre d'utilisateurs d'Internet et l'arrivée de nouveaux services et traitements multimédia ont eu comme conséquence l'augmentation des tailles des tables de routages des routeurs et la complexité de l'opération d'acheminement des paquets de données.

Il y a trois facteurs principaux à être considérés dans les réseaux IP pour maintenir un bon service: Liens avec une grande largeur de bande, commutation de données à grande vitesse et taux élevés de l'expédition de paquets.

Actuellement, les liens optiques de transmission de données et la technologie courante de commutation de données permettent la manipulation facile des deux premiers facteurs. Par conséquent, le déploiement des routeurs de rendement élevé pour expédier les paquets IP est la clef au succès des routeurs IP.

Dans l'Internet, chaque paquet est acheminé indépendamment des autres. Ce mode d'opération est connu comme le mode datagramme. Le mode datagramme permet d'offrir un service robuste, car les routeurs peuvent adapter l'acheminement des paquets lors des changements dans la topologie des réseaux. Cependant, le mode datagramme nécessite que les

routeurs aient la capacité suffisante pour traiter tous les paquets arrivant à leurs ports d'entrée. Ainsi, avec l'accroissement du trafic, il est nécessaire d'optimiser la performance des routeurs lors de l'acheminement des paquets.

Chaque routeur maintien une table de routage qui est construite à partir des informations échangées avec d'autres routeurs. Ces échanges d'information sont réalisés par l'intermédiaire de protocoles de routage, tels que RIP, OSPF ou BGP. Cette table de routage contient en général plus d'information que celle strictement nécessaire pour l'acheminement de paquets (comme les informations de gestion du réseau).

Pour acheminer les paquets, les routeurs doivent accomplir deux tâches principales :

Premièrement, les routeurs doivent déterminer où doit envoyer chaque paquet reçu. Plus spécifiquement, les routeurs doivent déterminer, pour chaque paquet reçu, l'adresse du prochain routeur (ou l'adresse de la destination finale s'il s'agit du dernier relais) et le port de sortie par lequel sera réexpédier le paquet. On appelle l'ensemble de ces deux informations les informations de routage. Pour déterminer les informations de routage, le routeur utilise l'adresse IP de destination du paquet reçu pour la recherche dans la table de routage. Cette opération s'appelle consultation d'adresse (*address lookup*) ou recherche d'information de routage.

Deuxièmement, les routeurs doivent commuter le paquet du port d'entrée au port de sortie approprié. Si le lien de sortie est disponible, le paquet sera retransmis sur ce lien ; dans le cas contraire, le routeur doit mémoriser le paquet dans un buffer en attendant que le lien soit disponible.

Dans cette thèse, nous nous focalisons sur la première tâche (décision de routage).

1.2. Protocole IP et adressage

1.2.1. Protocole IP

IP est l'acronyme de "Internet Protocol", il est défini dans la RFC 791 [1] et a été conçu en 1980 pour remplacer NCP (Network Control Protocol) le protocole de l'ARPAnet. Il représente le protocole réseau le plus répandu. Il permet de découper l'information à transmettre en paquets, de les adresser, de les transporter indépendamment les uns des autres et de recomposer le message initial à l'arrivée.

Le protocole IP fait partie de la couche Internet de la suite de protocoles TCP/IP, et permet l'élaboration et le transport des datagrammes IP, sans toutefois en assurer la livraison. Concrètement, le protocole traite les datagrammes IP indépendamment les uns des autres en définissant leur routage et leur expédition utilisant l'adresse IP de destination [1].

1.2.2. Evolution d'adressage IP

1.2.2.1. L'adressage avec classes

Une des tâches fondamentale lors de la mise en place d'un réseau TCP/IP consiste à affecter des adresses Internet aux nœuds du réseau. Ce sont les adresses IP. Si l'affectation des adresses IP peut sembler facile à première vue, il est nécessaire de prendre en considération un certain nombre de points. Pour que l'ensemble soit cohérent et puisse fonctionner, tous les périphériques doivent avoir une adresse unique. De plus, afin de pouvoir communiquer avec les autres nœuds, il est indispensable que ces adresses soient cohérentes entre elles.

La base de l'adressage dans un réseau TCP/IP est l'adresse de la carte réseau (adresse MAC). Elle est inscrite en dur dans le programme implémenté. Pour de nombreuses raisons (changement de carte réseau, facilité de groupage de mêmes types d'adresses), la décision fut prise pour mettre en place un système d'adressage logique indépendant au-dessus des adresses physiques.

1.2.2.1.1. Structure de l'adresse IP

La vision logique de l'Internet créé un réseau virtuel. Chaque connexion de réseau à ce réseau virtuel est identifiée de manière distincte au moyen d'une adresse IP. Les concepteurs de TCP/IP ont choisi un adressage sur 32 bits (IPv4) [2] et 128 bits pour (IPv6). Le réseau virtuel est constitué de réseaux reliés entre eux par le biais de périphériques tels que routeurs ou passerelles.

Afin d'aiguiller les datagrammes IP, les routeurs doivent être en mesure de distinguer différent réseaux logiques. L'adresse IP est structurée de façon à ce qu'elle puisse refléter la distinction entre les différents réseaux logiques. Un certain nombre de bits dans l'adresse IP sont utilisés pour identifier le réseau individuel dans le réseau virtuel, et les bits suivants permettent d'identifier l'hôte au sein du réseau.

32 bits

Netid	**Hostid**
N bits	32 – N bits

La méthode consistant à partager l'adresse IP entre un numéro de réseau et un numéro d'hôte constitue un plan d'adressage hiérarchique, ce qui permet de rendre le routage beaucoup plus efficace. La fonction première d'un routeur est d'envoyer un datagramme IP dans au réseau. A cet effet, les routeurs doivent stocker les informations concernant les NetIds et non les HostIds.

Le nombre de Netids est forcément inférieur aux Hostids, ce qui maintient à un nombre raisonnable la quantité d'informations qu'un routeur doit connaître. Si l'on n'avait pas établit cette distinction entre Netid et Hostid (en choisissant un système d'adressage plat, plutôt qu'un plan d'adressage hiérarchique), les routeurs auraient dû stocker les quarte milliards d'adresses IP.

A partir du nombre de 32 bits utilisé pour l'adresse IPv4 [2], les concepteurs ont décidé que l'on utiliserait le premier, les 2 premiers ou les 3 premiers octets comme Netid selon les classes d'adresses IP.

1.2.2.1.2. Classes d'adresses IP

Dans les premières années d'Internet, l'assignation des adresses aux réseaux finaux consistait à octroyer le premier octet de l'adresse au réseau, c'est-à-dire que 256 réseaux de 16 millions d'adresses étaient possibles. Devant la limitation qu'impose ce modèle, une modification de la façon dont les adresses sont assignées est indispensable. En 1981, la RFC 790 [3] (*Assigned numbers*) prévoit qu'une adresse IP est divisée en deux parties : une partie servant à identifier le réseau (net id) et une partie servant à identifier un poste sur ce réseau (host id).

Il existe 5 classes d'adresses IP. Chaque classe est identifiée par une lettre allant de A à E. Ces différentes classes ont chacune leurs spécificités en termes de répartition du nombre d'octet servant à identifier le réseau ou les ordinateurs connectés à ce réseau :

- Une adresse IP de classe A dispose d'une partie *NetId* comportant uniquement un seul octet.
- Une adresse IP de classe B dispose d'une partie *NetId* comportant deux octets.
- Une adresse IP de classe C dispose d'une partie *NetId* comportant trois octets.
- Les adresses IP de classes D et E correspondent à des adresses IP particulières.

Afin d'identifier à quelle classe appartient une adresse IP, il faut examiner les premiers bits de l'adresse.

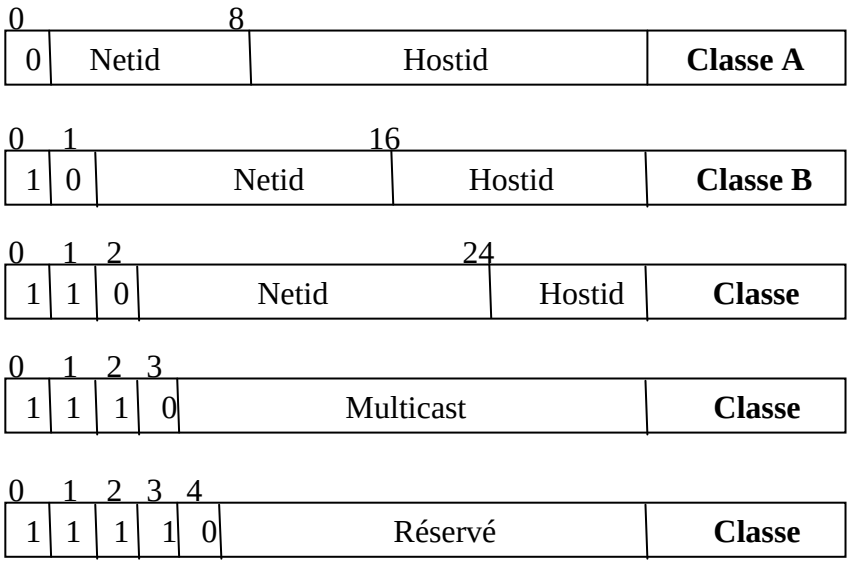


Figure.1.1. Classes d'adresses IP

Pourquoi utiliser des classes d'adresses spécifiques ?

Les différents types de classes d'adresses IP sont définis pour répondre aux besoins des réseaux de différentes tailles. Sur demande, l'autorité d'enregistrement du réseau affecte un numéro de réseau (NetId) à une organisation. Une fois ce numéro alloué à une organisation, il incombe à cette dernière d'affecter les numéros d'hôte (HostId).

1.2.2.1.3. Adresses IPv4

Ces adresses 32 bits correspondent au format d'adressage IP initialement conçu pour TCP/IP. À l'origine, les réseaux IP se décomposent en trois classes, A, B et C. Le numéro de réseau attribué à un réseau reflète cette identification de classe, tandis que les 8 bits ou plus supplémentaires représentent un hôte. Les adresses IPv4 basées sur les classes requièrent la configuration d'un masque de réseau pour le numéro de réseau. En outre, ces adresses étaient souvent divisées en sous-réseaux afin d'augmenter le nombre d'adresses disponibles pour les systèmes du réseau local.

Aujourd'hui, les adresses IP sont appelées adresses IPv4 ou IPv6. Il est désormais impossible d'obtenir des numéros de réseau IPv4 basés sur les classes auprès d'un fournisseur de services Internet, mais de nombreux réseaux les utilisent encore.

Limites d'IPv4

La version IPv4 a été en service pendant presque trois décennies. Mais, elle commence à montrer des limites vis-à-vis des demandes émergentes en termes de la cardinalité d'espace d'adressage, la mobilité à haute densité, le multimédia, et la forte sécurité. Par conséquent, l'IPv6 crée un nouveau format d'adresse IP, de sorte que le nombre d'adresse IP ne s'épuise pas pendant plusieurs décennies, ou bien aussi longtemps que possible à ce qu'une nouvelle récolte entière d'équipements soit reliée à l'Internet. IPv6 améliore également le routage et la configuration automatique de réseau.

1.2.2.1.4. Adresses IPv6

IPv6 est le protocole actuel d'Internet, qui était d'abord appelé IPng (Internet Next Generation). L'IETF [4] a développé les caractéristiques de base de ce protocole pendant les années 90 pour supporter la migration à un nouvel environnement. IPv6 défini dans RFC 2460 [5] et RFC 3513 [6] est considéré comme étant une version améliorée d'IP. Celui-ci est conçu pour coexister avec IPv4 et pour fournir par la suite de meilleures possibilités d'interconnexion de réseaux qu'avec IPv4 [2].

L'IETF a déployé des adresses IPv6 128 bits afin de résoudre à long terme le problème d'épuisement des adresses IPv4 disponibles. Les adresses IPv6 assurent un espace d'adressage plus étendu qu'IPv4 [7]. De même que les adresses IPv4 au format CIDR, les adresses IPv6 ne possèdent aucune notion de classe de réseau ni de masque de réseau. Comme dans CIDR, les adresses IPv6 utilisent des préfixes pour désigner la partie de l'adresse définissant le réseau du site.

Les avantages d'IPv6 peuvent être résumés comme suit :

- **Facteur d'échelle** (scalabilité) : les adresses IPv6 sont représentées sur 128 bits par contre celles d'IPv4 sont représentées sur 32 bits.
- **Sécurité** : IPv6 inclut la sécurité dans ses spécifications, tels que le chiffrement de données et l'authentification de la source de données.
- **Application en temps réel** : Pour fournir un meilleur support pour le trafic en temps réel (par exemple, VoIP), IPv6 inclut "des flux étiquetés" dans ses spécifications. Grâce à ce mécanisme, les routeurs peuvent identifier le flux de bout en bout auquel appartiennent les paquets transmis.
- **Adressage et routage** : IPv6 améliore l'adressage et le routage hiérarchique.
- **Extensibilité** : IPv6 a été conçu pour être extensible et offre un support pour de nouvelles options.

1.2.2.1.5. Sous-réseaux

Les formats d'adresses IPv4 ont étés conçus pour s'adapter à des réseaux de différentes tailles. Ces formats convenaient bien dans les premiers temps de l'internet. La principale faiblesse du format IPv4 réside dans le gâchis d'espace d'adressage consécutif à la conception d'un inter réseau.

Pour pallier à ce défaut, la RFC 950 [1] a défini et normalisé le concept de sous-réseau en 1985. Les sous-réseaux permettent d'utiliser un numéro de réseau unique pour construire plusieurs réseaux interconnectés. Le préfixe de numéro de réseau est partagé entre plusieurs réseaux appelés sous-réseaux. Cette mise en sous-réseaux est généralement utilisée lorsque le même numéro de réseau est utilisé pour plusieurs réseaux interconnectés.

Masque de sous-réseau

Outre l'adresse IP, une machine doit aussi connaître le nombre de bits attribués à l'identification du sous-réseau et à l'identificateur de machine. Ces informations sont fournies par le masque de sous-réseau (Netmask). Ce masque est un masque de 32 bits (pour IPv4) contenant soit des bits à 1 pour l'identification du réseau et des bits à 0 pour l'identification de machines.

1.2.2.2. Structuration des adresses et agrégation

Un des problèmes majeurs dans le routage est la croissance incontrôlée des tables de routage. Ce phénomène est dû à une mauvaise agrégation des adresses dans les tables. Il faudrait être capable de router des ensembles de réseaux identifiés par un seul descripteur. CIDR apporte une amélioration.

1.2.2.2.1. Evolution des tailles des tables de routage

Actuellement, pour pouvoir assurer une bonne agrégation, les règles utilisées par CIDR [8] pour IPv4 sont conservées, mais la gestion des tables de routage dans le cœur du réseau s'en trouvera quand même améliorée car :

- Dès le début, le plan d'adressage est hiérarchique et éliminant les longs préfixes,
- Les sites multi-domiciliés posséderont autant d'adresses que de fournisseurs, permettant ainsi de garantir une agrégation.
- Des mécanismes de renumérotation automatiques permettent aux sites de changer facilement de préfixe quand cela est nécessaire.

La figure suivante présente l'évolution des tables de routage dans le cœur de l'Internet.

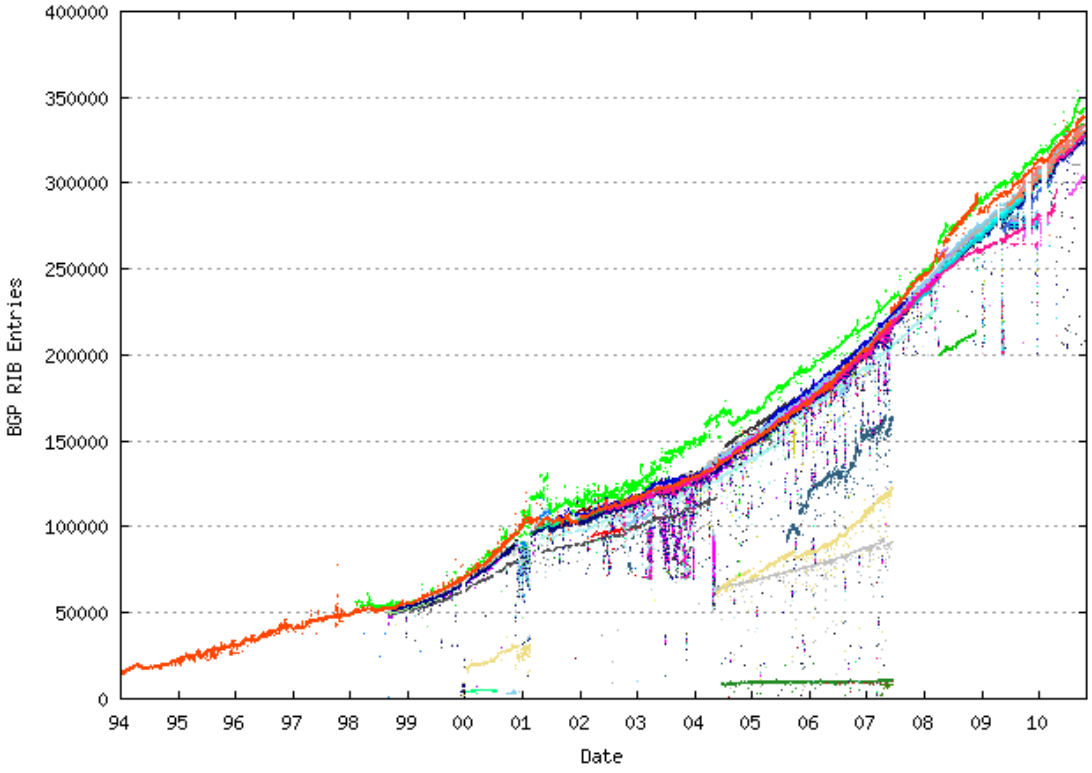


Figure.1.2. Evolution des tailles des tables de routage [8]

1.2.2.2.2. CIDR (Classless Inter-Domain Routing)

En 1992, la RFC 1338 [9] **(***Supernetting: an Address Assignment and Aggregation Strategy*) propose de détruire la notion de classe qui n'était plus adaptée à la taille d'Internet. Le CIDR (*Classless Inter-Domain Routing*) [10], [8] est mis au point en 1993, afin de diminuer la taille de la table de routage contenue dans les routeurs. Ce but est atteint en agrégeant plusieurs entrées de cette table en une seule.

Le but était de pouvoir regrouper plusieurs réseaux de classe C dans un seul bloc d'adresses de 2^n x 256 afin de poser une seule entrée vers ces réseaux (agrégation de routes), on parlait alors de supernetting, ceci fut ensuite propagé aux adresses de classe B bien que le besoin d'agrégation y soit moindre, puis enfin aux réseaux de classe A, bien que le problème d'agrégation des routes ne s'y pose pas.

En fait c'est toute la représentation de l'espace d'adressage qui a été changé.

Principe

La conception initiale de sous-réseaux par classe nécessitait que tous les sous-réseaux d'un même réseau classé soient de la même taille. Cela découlait du fait que les routeurs n'incluaient pas de données de masque de sous-réseau dans leurs mises à jour de routage. Un routeur programmé avec une adresse et un masque de réseau sur une interface appliquait automatiquement le même masque aux autres sous-réseaux dans sa table de routage. Du fait de cette limite, le schéma d'adressage IP nécessitait la planification de masques de sous-réseau de longueur fixe.

Cependant, les masques de sous-réseau de longueur fixe peuvent gaspiller un nombre important d'adresses IP. Par exemple, prenons une organisation comprenant un site doté d'environ 8 000 hôtes et trois autres emplacements dotés respectivement de 1 000, 400 et 100 hôtes. Avec un masque de sous-réseau de longueur fixe, chaque sous-réseau devrait prendre en charge au moins 8 000 hôtes, y compris le sous-réseau affecté à l'emplacement qui ne nécessite que 100 adresses.

Le masquage de sous-réseau de longueur variable (VLSM) permet de résoudre ce problème. L'adressage VLSM permet de diviser un espace d'adressage en réseaux de tailles différentes. Cette opération s'effectue en découpant les sous-réseaux. Pour ce faire, les routeurs doivent recevoir des informations de routage incluant l'adresse IP du réseau, ainsi que des données de masque de sous-réseau indiquant le nombre de bits qui composent la partie réseau de l'adresse IP. Le masquage de sous-réseau de longueur variable permet d'économiser des milliers d'adresses IP que la création traditionnelle de sous-réseaux par classe aurait gaspillée.

En plus du masquage de sous-réseau de longueur variable (VLSM), le document RFC1519 [11] a proposé le routage inter-domaine sans classe CIDR (*Classless Inter-Domain Routing*), lequel a été accepté. Le routage inter-domaine sans classe (CIDR) ignore les classes de réseau basées sur la valeur des bits d'ordre haut. Il identifie les réseaux uniquement d'après le nombre de bits du préfixe réseau, lequel correspond au nombre de 1 dans le masque de sous-réseau.

L'adresse 172.16.1.1/16 est un exemple d'adresse IP écrite à l'aide de la notation CIDR dans laquelle /16 représente le nombre de bits du préfixe réseau.

Les protocoles CIDR ont donné aux routeurs la capacité de déterminer le préfixe réseau sans être limités aux bits d'ordre haut. La suppression de cette restriction a éliminé la nécessité d'attribuer les adresses IP enregistrées par classes d'adresses.

Avant l'utilisation du protocole CIDR, un FAI nécessitant 3 000 adresses d'hôte pouvait demander soit un espace d'adressage de classe B complet, soit plusieurs adresses réseau de classe C pour répondre à ses besoins. Avec un espace d'adressage de classe B, le FAI aurait gaspillé des milliers d'adresses enregistrées. Avec plusieurs adresses de classe C, il pouvait être difficile de concevoir le réseau FAI de sorte qu'aucune section unique ne nécessite plus de 254 adresses d'hôte. Les tables de routage contenant de nombreuses adresses de classe C peuvent également devenir trop volumineuses et difficiles à gérer.

Du fait qu'il ignore les classes d'adresses traditionnelles, le protocole CIDR permet au FAI de demander un bloc d'adresses qui soit fonction du nombre d'adresses d'hôte requis. Les super-réseaux, créés en regroupant un groupe d'adresses de classe C dans un bloc de grande taille, permettent une attribution plus efficace des adresses. Un exemple de super-réseau pourrait être 192.168.0.0/19. L'utilisation des 19 premiers bits de l'adresse IP pour le préfix réseau permet à ce super-réseau de contenir 8190 adresses d'hôte possibles. Un FAI peut utiliser un super-réseau comme un grand réseau unique ou le diviser en autant de réseaux plus petits que nécessaire pour répondre à ses besoins.

Dans cet exemple de super-réseau, l'adresse de classe C privée 192.168.0.0 est utilisée. En réalité, la plupart des réseaux utilisant l'adressage privé utilisent des adresses réservées et le découpage en sous-réseaux de classe A ou B. Bien que l'adressage classé et le masquage de sous-réseau de longueur fixe deviennent moins courants, il est important de comprendre le fonctionnement des méthodes d'adressage. De nombreux périphériques utilisent encore le masque de sous-réseau par défaut si aucun masque de sous-réseau personnalisé n'est défini.

Communications entre sous-réseaux

Lorsqu'un réseau est divisé en sous-réseaux, chacun de ces sous-réseaux constitue réellement un réseau à part entière. Les routeurs ayant pour fonction de connecter les réseaux, un routeur est donc indispensable pour qu'un périphérique d'un sous-réseau puisse communiquer avec un périphérique d'un autre sous-réseau.

Pour déterminer le nombre d'hôtes requis dans chaque sous-réseau, il est nécessaire d'inclure l'interface de routeur, ou l'interface de passerelle, ainsi que les périphériques hôte individuels.

Chaque interface de routeur doit avoir une adresse IP dans le même sous-réseau que le réseau d'hôtes qui lui est lié.

Adresses IPv4 au format CIDR

L'IETF [4] a développé des adresses CIDR *(Classless Inter-Domain Routing)*, routage inter-domaine sans classe, dans le but de résoudre à court ou moyen terme le problème d'épuisement des adresses IPv4 [2]. Par ailleurs, le format CIDR a été conçu pour remédier au manque de capacité des tables de routage Internet. Une adresse IPv4 avec notation CIDR présente une longueur de 32 bits et le même format décimal avec points. Cependant, CIDR

ajoute une désignation de préfixe juste après l'octet le plus à droite afin de définir la portion de réseau de l'adresse IPv4.

1.3. Recherche d'information de routage dans un routeur IP

1.3.1. Routeurs IP

Les routeurs sont les dispositifs permettant de "choisir" le chemin que les datagrammes vont emprunter pour arriver à la destination. Il s'agit de machines ayant plusieurs ports d'entrée dont chacun est relié à un réseau différent. Ainsi, dans la configuration la plus simple, le routeur n'a qu'à "regarder" sur quel réseau se trouve un ordinateur pour lui faire parvenir les datagrammes en provenance de l'expéditeur.

Toutefois, sur Internet le schéma est beaucoup plus compliqué pour les raisons suivantes:

- Le nombre de réseau auxquels un routeur est connecté est généralement important.
- Les réseaux auxquels le routeur est relié peuvent être reliés à d'autres réseaux que le routeur ne connaît pas directement.

Ainsi, les routeurs fonctionnent grâce à des tables de routage et des protocoles de routage, selon le modèle suivant:

- Le routeur reçoit une trame provenant d'une machine connectée à un des réseaux auquel il est rattaché.
- Le routeur regarde l'entête du datagramme, si l'adresse IP de destination appartient à l'un des réseaux auxquels une des interfaces du routeur est rattaché, l'information doit être envoyée à la couche supérieure après que l'entête IP ait été désencapsulée (enlevée), dans ce cas on parle de remise directe.
- Si l'adresse IP de destination fait partie d'un réseau différent, le routeur consulte sa table de routage, une table qui définit le chemin à emprunter pour une adresse donnée, le routeur envoie le datagramme grâce au port de sortie relié au réseau sur lequel le routeur décide d'envoyer le paquet, on parle de remise indirecte.

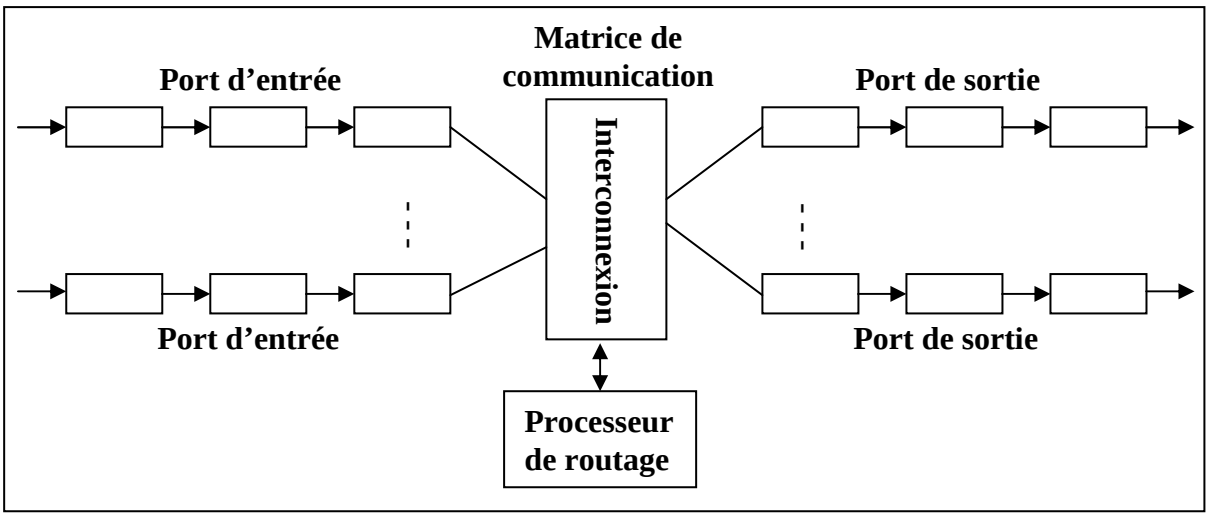


Figure.1.3. Architecture d'un routeur Internet

1.3.2. Table de routage

La table de routage est une table de correspondance entre l'adresse de la machine ou le préfixe du sou-réseau visée et le nœud suivant auquel le routeur doit délivrer le message, elle peut être entrée manuellement par l'administrateur (*routage statique*) ou construite par le routeur en fonctions des informations qu'il reçoit (*routage dynamique*). En réalité il n'est pas nécessaire de stocker l'adresse IP complète de la machine: Seul l'identificateur du réseau de l'adresse IP est stocké.

La table de routage est donc un tableau contenant des paires d'adresses:

Adresse de destination	Adresse du prochain routeur directement accessible	Interface

Ainsi grâce à cette table, le routeur, connaissant l'adresse du destinataire encapsulé dans le message, va être capable de savoir sur quelle interface envoyer le message (cela revient à savoir quelle carte réseau utiliser), et à quel routeur, directement accessible sur le réseau auquel cette carte est connectée, remettre le datagramme.

Ce mécanisme consistant à ne connaître que l'adresse du prochain maillon menant à la destination est appelé routage par sauts successifs (*next-hop routing*).

Le message est ainsi remis de routeur en routeur par sauts successifs, jusqu'à ce que le destinataire appartienne à un réseau directement connecté à un routeur. Celui-ci remet alors directement le message à la machine visée.

Dans les premiers grands réseaux, les tables de routage étaient statiques et donc maintenues à jour par des techniciens de bout en bout. De nos jours, les mises à jour des tables de routage et le calcul du meilleur chemin sont automatiquement propagés sur le réseau par les protocoles de routage.

1.3.3. Consultation d'adresses IP (Notre contribution)

Pour acheminer les paquets, les routeurs doivent déterminer où doit envoyer chaque paquet reçu. Plus spécifiquement, les routeurs doivent déterminer, pour chaque paquet reçu, l'adresse du prochain routeur (ou l'adresse de la destination finale s'il s'agit du dernière relais) et le port de sortie par lequel sera réexpédié le paquet. On appelle l'ensemble de ces deux informations l'information de routage.

Pour déterminer l'information de routage, le routeur consulte l'adresse de destination du paquet reçu dans une table de routage. Cette opération s'appelle consultation d'adresse (*address lookup*).

1.3.3.1. Consultation d'adresses IP dans le cas d'adressage en classes

L'architecture d'adressage en classe basée sur des routeurs permet d'utiliser une opération de recherche relativement simple. En générale, la table de routage avait trois parties, une pour chacune des trois classes unicast A, B et C. Les entrées de la table de routage sont des tuples de la forme « *Netid*, adresse de *Next-hop* ». Toutes les entrées de la même partie *Netid* sont de largeur fixe : 7, 14 et 21bits respectivement pour les classes A, B et C, et l'opération de recherche pour chaque paquet entrant a procédé comme dans la figure.1.4. Tout d'abord, la classe a été déterminée à partir des bits les plus significatifs de l'adresse de destination du paquet. Ceci à son tour de déterminer lequel des trois parties de la table de routage à utiliser, en suite le routeur effectuer une recherche pour une correspondance exacte entre le *Netid* du paquet entrant et une entrée dans la partie sélectionnée de la table de routage.

La recherche avec correspondance exacte peut être effectuée en utilisant, par exemple, le hachage ou un algorithme de recherche binaire.

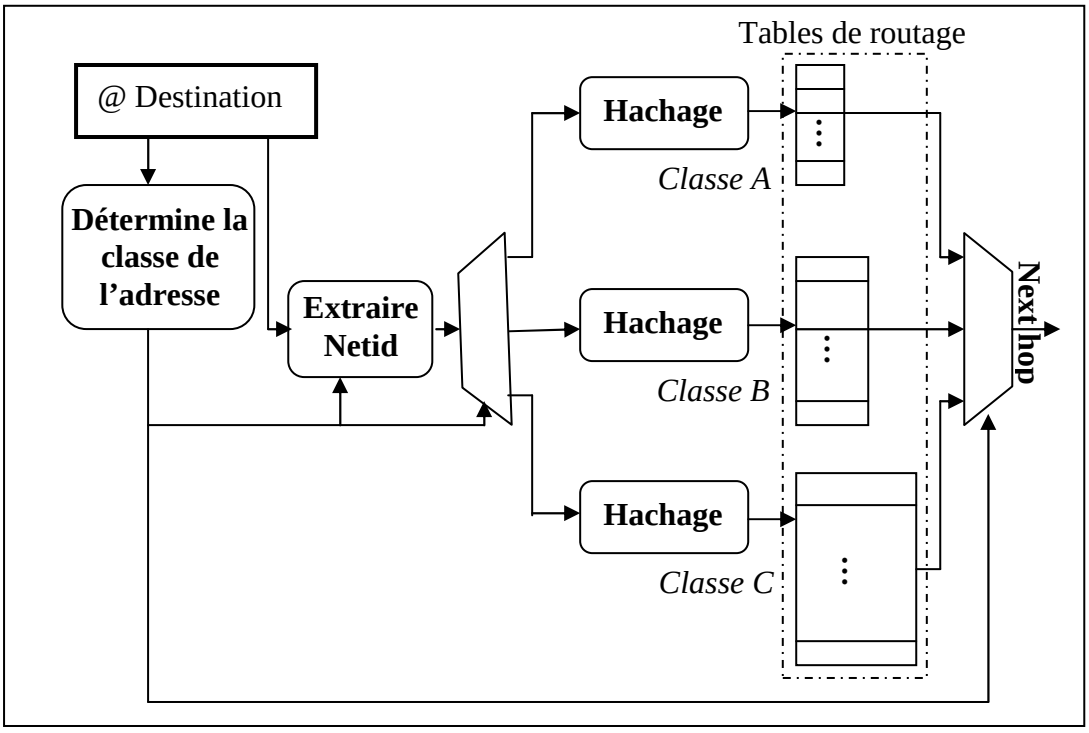


Figure.1.4. Implémentation de l'opération de lookup dans le cas d'adressage en classes

L’dressage en classes a bien fonctionné dans les premiers jours de l'Internet mais, à cause de la croissance d'Internet, deux problèmes ont survenu : Une insuffisance de l’espace d’adressage IP et une croissance exponentielle des tailles des tables de routage ce qui complique l’opération de recherche dans ces tables de routage.

Le problème de tailles des tables de routages est dû au fait que le routeur IP doit stocker toutes les adresses *Netid* dans sa table de routage. En conséquence, les tables de routage sont en croissance exponentielle. Ceci a placé une charge élevée sur les ressources du processeur et la mémoire des routeurs de l'Internet.

Dans une tentative pour ralentir la croissance des tailles des tables de routage et de permettre une utilisation plus efficace de l'espace d'adressage IP, un nouveau schéma d’adressage et de routage appelé CIDR (*Classless Inter-Domain Routing*) a été adopté en 1993 [11] [10].

1.3.3.2. Consultation d’adresses IP dans le cas d’adressage sans-classes (CIDR)

Pour pouvoir passer à l'échelle, les tables de routage n'ont pas une entrée pour chaque adresse destination, mais une entrée par groupes d'adresses. En particulier, les adresses sont agrégées en utilisant leurs préfixes communs. Ainsi, chaque entrée de la table de routage contient l'information de routage correspondant à un groupe d'adresses destination représenté par le préfixe commun de ces adresses. Historiquement, il y avait trois tailles fixes de préfixes (8, 16 et 24 bits). Chaque taille fixe de préfixe déterminait une classe différente d'adresses. Mais à l'heure actuelle, la taille des préfixes peut être de 0 à 32 bits pour le protocole IPv4 et de 0 à 128 bits pour IPv6 dans ce que l'on appelle le routage inter domaines sans classes (CIDR).

Une des opérations essentielles pour acheminer les paquets arrivant à un routeur est la décision de routage, c'est-à-dire la recherche d'information de routage dans la table de routage (plus long préfixe correspondant). Depuis l'introduction de CIDR (*Classless Interdomain Routing*), cette recherche d'information de routage consiste à trouver l'entrée de la table de routage ayant le plus long préfixe commun avec l'adresse destination du paquet à acheminer. En effet, plusieurs entrées de la table de routage peuvent avoir un préfixe commun avec l'adresse destination, mais l'entrée avec le plus long préfixe aura l'information de routage la plus spécifique et donc c'est cette entrée qui doit être utilisée pour effectuer l'acheminement du paquet.

Avec CIDR, La table de routage d'un routeur se compose d'entrées de la forme « préfixe, Next-Hop », où le préfixe est un préfixe IP et Next-Hop est l'adresse IP du prochain saut. Une adresse IP de destination correspond à un préfixe si les bits significatifs du préfixe sont identiques aux premiers bits de l'adresse IP de destination. L’opération de recherche d’information de routage pour un paquet entrant exige de trouver le préfixe le plus spécifique pour le paquet. Cela implique que le routeur a besoin de résoudre le problème de plus long préfixe correspondant à l’dresse IP du paquet *(longest prefix matching problem),* défini comme suit :

Définition.1.1 : Le problème de recherche du plus long préfixe correspondant à une adresse IP est le problème de recherche de l'entrée de la table de routage contenant le plus long préfixe parmi tous les préfixes (dans d'autres entrées de la table de routage) correspondant l'adresse de destination du paquet entrant. Ce plus long préfixe est appelé le plus long préfixe correspondant (*longest prefix matching)*.

Exemple.1.1 : Prenant comme exemple un routeur contenant une table de routage représentée dans le tableau.1.1. Si un paquet entrant à ce routeur a une adresse de destination « 200.11.0.1 », elle corresponde les deux préfixes 200.11.0.0/22 et 200.11.1.1/20 mais le préfixe 200.11.0.0/22 est plus long que le préfixe 200.11.1.1/20, alors le paquet sera transmis à l'interface de sortie Nh_2 du routeur.

Préfixe	Next-Hop
192.2.0.0/22	Nh1
200.11.0.0/22	Nh2
192.2.2.0/24	Nh3
200.11.1.1/20	Nh4
192.2.3.0 /18	Nh4
200.10.0.0/16	Nh1
192.1.1.0/10	Nh2
200.3.2.0/18	Nh3

Tableau.1.1. Exemple 1.1 de préfixes

1.4. Difficulté de recherche d'information de routage

Afin d'augmenter l'efficacité du routage dans l'Internet, plusieurs adresses destination sont agrégées en une seule entrée dans une table de routage. Au départ, cette agrégation correspondait au regroupement de toutes les adresses appartenant à un même réseau. En conséquence le routage était effectué en considérant uniquement les réseaux et non pas chaque hôte en particulier. En raison de la façon dont les adresses sont allouées, toutes les adresses appartenant à un même réseau ont le même préfixe**.** La recherche dans la table de routage peut s'effectuer en comparant les préfixes avec les bits correspondants de l'adresse IP de destination. Avec cette agrégation d'adresses, la taille des tables de routage n'augmente pas avec le nombre de hôtes, cependant le nombre d'entrées dans la table augmente toujours avec le nombre de réseaux.

Depuis l'apparition du système d'adressage CIDR [10], [8] les difficultés d'expédition des paquets IP sont à cause de :

- Les longueurs du préfixes sont variables et l'adresse du paquet de données ne donne pas une information sur la longueur du préfixe à consulter;
- Une adresse IP peut correspondre à plusieurs préfixes dans la table de routage et le plus long préfixe correspondant devrait être choisi.

L'adresse IP de destination d'un paquet de données ne contient pas les informations nécessaires pour déterminer la longueur du préfixe à consulter (le plus long préfixe correspondant cette adresse). Par conséquent, nous ne pouvons pas trouver la correspondance en utilisant un algorithme de recherche de correspondance exacte (par exemple, en utilisant le hachage ou bien une procédure de recherche binaire). Au lieu de cela, une recherche du plus long préfixe correspondant doit déterminer à la fois la longueur du plus long préfixe correspondant ainsi que l'entrée de la table de routage contenant le préfixe de cette longueur qui correspond l'adresse de destination du paquet entrant.

1.5. Contributions

Cette thèse a pour but l'étude du problème de recherche d'information de routage au niveau des tables de routage des routeurs Internet.

Les contributions de nos recherches sont multiples :

- L'étude bibliographique approfondie (chapitre 2) nous a permis de : Synthétiser les concepts relatifs aux tâches des routeurs IP, et de dégager la problématique sur laquelle nous avons travaillé, et de faire une classification des solutions proposées pour notre problématique qui est la recherche d'information de routage au niveau d'une table de routage d'un routeur IP, vue la croissance exponentielle des tailles des tables de routage.
- Proposition des solutions pour notre problématique (Chapitre 4 et 5). Compte tenu que le but principal d'un routeur est d'acheminer des paquets, un facteur capital dans la performance d'un routeur est la vitesse avec laquelle le routeur trouve les informations de routage. Une façon d'optimiser le temps pour trouver le plus long préfixe, et donc de trouver l'information de routage, et d'utiliser un mécanisme efficace de recherche du plus long préfixe. Dans cette thèse nous proposons des mécanismes de recherche du plus long préfixe correspondant à une adresse IP dans une table de routage, les mécanismes que nous avons proposés dans cette thèse contribuent à améliorer la performance de l'acheminement de paquets dans les routeurs IP.

Pour les solutions proposées, la démarche suivie est :

Proposition d'une solution matérielle :

Algorithme et architecture parallèle dédiée. Nous proposons une architecture parallèle appelée « PARIR » (***P**arallel **A**rchitecture for **R**outing **I**nformation **R**esearch*) et ce pour la recherche d'information de routage (*Next-hop*) ou bien le port de sortie pour un paquet de données. À la différence d'autres approches notre mécanisme basé sur le critère de longueur

des préfixes pour regrouper les préfixes dans des sous tables de routages équilibrés et manipulés en parallèle par une architecture parallèle dédiée afin d'accélérer l'opération de recherche du plus long préfixe pour une adresse IP d'un paquet de données reçue d'un routeur.

Une modélisation et description VHDL de cette architecture nous permettent de tester par simulation les différents modules de notre architecture.

Proposition d'une solution logicielle :

Elle consiste en un algorithme avec table de routage cache. Nous proposons une technique de recherche d'information de routage basée sur une table de routage cache qui mémorise les adresses IP récemment consultées et leurs informations de routage pour augmenter la vitesse de recherche d'information de routage dans un routeur IP, le routeur extraira l'adresse IP de destination d'un paquet de données et cherche l'interface de sortie correspondante dans la table de routage cache, Si cette information n'existe pas dans la table de routage cache une nouvelle recherche est déclenchée dans la table de routage principale utilisant la technique classique de recherche par préfixes.

Proposition d'une autre solution logicielle :

Elle consiste en un algorithme avec arbre binaire à contenu dynamique. Un nouvel algorithme de recherche de plus long préfixe en utilisant un arbre binaire à contenu dynamique (ABCD) est proposé. Le contenu de cet arbre est variable selon les adresses IP traitées par le routeur. La technique proposée sert à exploiter les nœuds vides de l'arbre binaire de préfixes [12] par le placement des copies de préfixes les plus récemment utilisés dans les nœuds vides des niveaux supérieurs de l'arbre, de tel sorte que l'opération de recherche du plus long préfixe se termine dès que le plus long préfixe correspondant à l'adresse IP du paquet de données est trouvée dans un certain nœud interne dans l'arbre binaire sans arriver obligatoirement à une feuille.

Étude de performance :

Des modèles de simulation sont développés pour les tests de nos solutions proposées. Plusieurs scénarios et paramètres de simulation sont considérés. Les paramètres utilisés reflètent exactement les caractéristiques du problème de recherche de plus long préfixe dans une table de routage. Les résultats sont analysés en se basant sur une étude comparative par rapport aux autres propositions.

Ces différentes propositions ont été validées par la communauté scientifique et ont donné lieu à deux publications dans des revues internationales et à six communications dans des conférences internationales.

1.6. Plan de la thèse

Ce document comprend sept chapitres. Le premier chapitre est consacré à l'introduction générale de la thèse, ce chapitre permet au lecteur d'acquérir les bases nécessaires pour la compréhension de la suite du document. Nous y ferons une présentation du domaine de routage et d'adressage dans l'Internet. Enfin, nous présentons en détaille le problème de recherche d'information de routage au niveau d'un routeur Internet.

Le chapitre 2 présente une classification des différentes solutions proposées ultérieurement. La plupart des solutions peuvent être classées dans des approches logicielles basées sur des structures arborescentes qui sont simples et faciles à implémenter et des approches matérielles, à savoir, les circuits ASIC, les mémoires associatives CAMs et TCAMs, les mémoires caches et la manipulation parallèle des tables de routage.

Nous rappelons les fondements du parallélisme et l'essentiel des architectures parallèles dans le chapitre 3.

Dans les chapitre 4 et 5 nous exposerons en détail les trois solutions que nous avons élaborées au cours de cette thèse afin d'améliorer les performances de l'opération de recherche d'information de routage dans les routeurs IP. La première (chapitre 4) est une solution matérielle basée sur une architecture parallèle, tandis que les deux dernières propositions (dans le chapitre 5) sont des solutions logicielle ; une proposition de mise en table cache des adresses IP récemment consultées et une autre solution algorithmique basée sur un arbre à contenu dynamique. Pour chacune de ces solutions une évaluation des performances et des comparaisons avec d'autres techniques sont faites.

Enfin, le Chapitre 6 termine nos travaux et propose différentes perspectives de recherche.

1.7. Conclusion

Une des opérations déterminantes de la performance globale d'un routeur est la recherche d'information de routage dans la table de routage pour router un paquet IP entrant au routeur. Pendant longtemps, le principal goulot d'étranglement dans les performances des routeurs IP a été le temps qu'il faut pour rechercher une route dans la table de routage. Le problème est défini comme étant celui de la recherche par le biais d'une base de données (table de routage) de préfixes et de localiser la destination (port de sortie) correspondant au plus long préfixe correspond à l'adresse de destination d'un paquet entré au routeur.

Nous avons vu dans ce chapitre une introduction et présentation du domaine de travail de cette thèse qui est la recherche d'information de routage dans un routeur Internet.

Dans le chapitre suivant, nous allons présenter un état de l'art des différentes solutions proposées pour améliorer les performances de l'opération de recherche d'information de routage dans les tables de routage des routeurs Internet.

Chapitre 2

Travaux dans le domaine

Avant de présenter nos propositions pour le problème de recherche d'information de routage dans les tables de routage des routeurs IP, nous avons présenté dans ce chapitre une classification des solutions trouvées dans la littérature.

Nous avons classé ces solutions en deux grandes classes : solutions logicielles et solutions matérielles. Pour les solutions logicielles, nous distinguons les algorithmes de recherche linéaire et les algorithmes de recherche. Les solutions matérielles sont classées en mémoire CAMs, mémoires caches et architectures multiprocesseurs.

2.1. Introduction

L'une des opérations essentielles pour acheminer les paquets arrivant à un routeur consiste en la décision de routage, c'est à dire la recherche d'information de routage dans la table de routage. La décision de routage permet de chercher la meilleure route pour un paquet à partir de son adresse IP de destination. Comme nous l'avons présenté dans le chapitre 1, avec le système d'adressage sans classes CIDR, l'opération de recherche d'information de routage requiert la recherche du plus long préfixe correspondant à une adresse IP.

Plusieurs solutions pour ce problème sont proposées dans la littérature. Cependant dans ce chapitre, nous proposons une classification de ces différentes solutions comme suit : Des approches logicielles, qui sont des algorithmes utilisant des structures arborescentes pour la représentation des préfixes de la table de routage, ces algorithmes sont classés selon le principe de représentation des préfixes dans l'arbre, à savoir, représentation binaire des préfixes (l'arbre stocke les bits des préfixes), la transformation et la représentation des préfixes sous forme de plages d'adresses IP, et représentation des valeurs des préfixes. Des approches matérielles, à savoir, les mémoires associatives CAMs et TCAMs, les mémoires caches et la manipulation parallèle des tables de routage.

2.2. Solutions logicielles et structures de données utilisées

2.2.1. Recherche linéaire

La structure de données la plus simple est une liste linéaire chainée de tous les préfixes dans la table de routage. L'algorithme de consultation traverse la liste de préfixes un par un, et donne le plus long préfixe assorti à la fin du parcours. La complexité de stockage de cette structure de données pour N préfixes est O(N). L'algorithme de consultation a une complexité de O(N). Les algorithmes d'insertion et de suppression ont une complexité de O(1), en supposant que l'endroit du préfixe à supprimer est connu.

Le seul avantage de cette approche est qu'elle utilise la mémoire d'une manière efficace, elle n'a pas besoin de stockage d'informations supplémentaires sauf les préfixes eux mêmes. L'opération d''insertion est simple, mais la suppression a besoin d'une opération de recherche.

2.2.1.1. Recherche linéaire basée sur les longueurs en utilisant les tables de hachage

Pour organiser les préfixes par longueur, on peut utiliser plusieurs tables différentes pour chaque longueur des préfixes. Dans chaque table, nous pouvons chercher une correspondance entre l'adresse IP et les préfixes à l'aide de hachage. Nous avons besoin de chercher le plus long préfixe correspondant à l'adresse IP, alors, l'approche la plus simple est celle qui consiste à effectuer une recherche séquentielle de la plus longue longueur vers la plus courte. Si nous supposons une fonction de hachage parfaite, alors la complexité du temps de recherche est en O(W), où W est la longueur maximale possible des préfixes.

2.2.2. Algorithmes de recherche binaires basés sur des structures arborescentes

2.2.2.1. Algorithmes de recherche binaires basés sur les bits des préfixes

A. Arbre binaire classique

Les routeurs agrègent l'information de routage utilisant des préfixes. Ainsi, dans une table de routage, chaque entrée contient un préfixe et son information de routage.

Les préfixes sont des chaînes de bits de longueur variable; ils peuvent être représentés tout naturellement par un trie [13]. Un trie est une structure de données en arbre qui organise ses données en tirant profit du caractère décomposable de ses données. Dans le cas spécifique des préfixes, ce sont les bits des préfixes qui sont utilisés pour déterminer les branches du trie. Par exemple, la figure.2.1 montre un trie binaire (chaque nœud a un maximum de deux fils) qui représente un ensemble de préfixes d'une table de routage. Les préfixes eux mêmes sont représentés par certains nœuds du trie : Chaque feuille du trie représente un préfixe ; mais les nœuds internes peuvent aussi représenter des préfixes.

Pour une adresse de destination donnée, la recherche du plus long préfixe dans un trie consiste essentiellement à parcourir le trie à partir de sa racine. On parcourt le trie en utilisant les bits de l'adresse de destination pour parcourir les branches correspondantes. Ainsi, à chaque nœud, la recherche se poursuivra à droite ou à gauche en fonction de la valeur du bit considéré. La recherche se termine lorsqu'il n'y a plus de chemins à suivre, et le dernier préfixe visité sera le plus long préfixe correspondant à l'adresse de destination donnée. La mise à jour d'un trie binaire est relativement facile. Néanmoins, le principal problème avec les tries binaires est que le nombre d'accès à la mémoire lors des recherches est grand. En effet, lors d'une recherche, à chaque fois que l'on teste un bit pour décider quelle branche emprunter dans le trie, un accès à la mémoire est nécessaire. C'est à dire que dans le pire des cas, une recherche a besoin de 32 accès à la mémoire dans IPv4, (128 accès à la mémoire dans IPv6).

Comme ces accès à la mémoire sont lents, la recherche dans un trie binaire n'est pas appropriée pour des routeurs de haute performance.

Préfixe	Longueur	Next-hop
00*	2	A
010*	3	B
101*	3	C
1*	1	D
111*	3	E
111111*	6	F
110100*	6	G
110101*	6	H
111100*	6	I
1100*	4	J

Tableau.2.1. Exemple 2.1 de préfixes

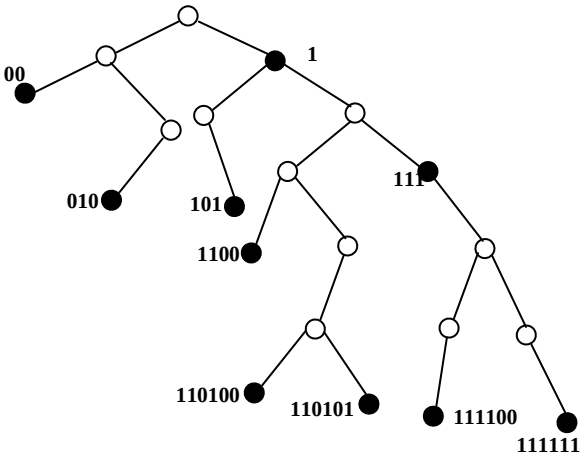


Figure.2.1. Arbre binaire construit à partir du tableau.2.1

B. Tri multi-bits

Une façon de réduire le nombre d'accès à la mémoire et d'optimiser le temps nécessaire pour trouver le plus long préfixe est l'utilisation de tries multi-bits. Dans un trie multi-bits, on ne parcourt pas le trie en testant un bit de l'adresse de destination à la fois, mais plusieurs bits à la fois. Plusieurs méthodes basées sur les tries multi-bits ont été proposées récemment [14], [15], [16], [17], [18], [19].

Un exemple de trie multi-bits est représenté dans la figure.2.2. Le parcours dans un trie multi-bits est effectué par des pas de plusieurs bits, alors, le trie multi-bits ne peut pas accepter des préfixes de longueur arbitraire. En effet, un trie multi-bits donné n'accepte que les préfixes de longueur déterminée par la taille des pas du trie multi-bits. Il est possible cependant d'utiliser un trie multi-bits pour représenter une table de routage quelconque. Pour ce faire, l'ensemble de préfixes de la table de routage doit être transformé en un autre ensemble de préfixes dont les longueurs soient acceptées par le trie multi-bits, tout en conservant la même information de routage. Cette transformation est réalisée par une technique appelée expansion de préfixes.

Bien que l'utilisation d'un trie multi-bits permet de réduire le nombre d'accès à la mémoire lors d'une recherche et donc d'améliorer la performance de la recherche d'information de routage, le fait de transformer l'ensemble original de préfixes rend plus difficile les opérations de mise à jour. Alors, la recherche dans un trie multi-bits est plus rapide, mais la mise à jour est beaucoup plus compliquée, par rapport au trie binaire [20].

Id	Préfixe
a	0*
b	01000*
c	011*
d	1*
e	100*
f	1100*
g	1101*
h	1110
i	1111

Tableau.2.2. Exemple 2.2 de préfixes

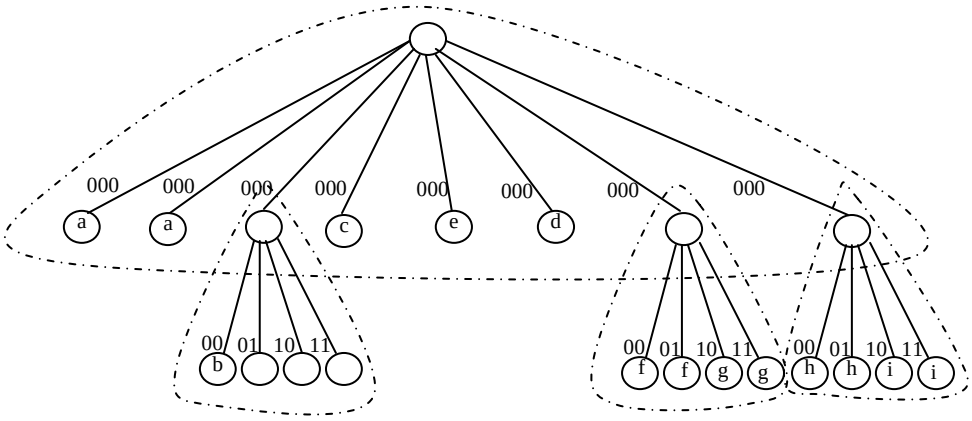


Figure.2.2. Trie multi-bits à pas fixe

Implémentation du trie multi-bits sur le matériel

Le schéma de base dans [14] emploie un trie multi-bits de 2 niveaux avec des pas fixes semblables à celui montré dans la figure.2.2 représentant les préfixes du tableau.2.2. Cependant, le premier niveau correspond à un pas de 24 bits et le deuxième niveau à un pas de 8 bits.

La plupart des entrées sont des préfixes de longueur de 24-bits ou inférieur. En conséquence, un pas de 24 bits nous a permet de trouver le plus long préfixe par un seul accès mémoire pour la majorité des cas. Afin de sauvegarder les préfixes dans la mémoire, on ne peut pas utiliser les nœuds internes de l'arbre binaire pour stocker les préfixes. Par conséquent, les préfixes correspondant à un nœud interne seront expansés au deuxième niveau (feuille de l'arbre). Pour l'exemple de la figure.2.2. Le premier niveau du trie multi-bits a 2^{24} nœuds, ceci est implémenté par un tableau de 2^{24} entrées. Une entrée au premier niveau contient l'information de routage ou un pointeur vers un sous-arbre correspondant au deuxième niveau. Le nombre de sous arbre du deuxième niveau dépend du nombre de préfixes de longueur supérieur à 24.

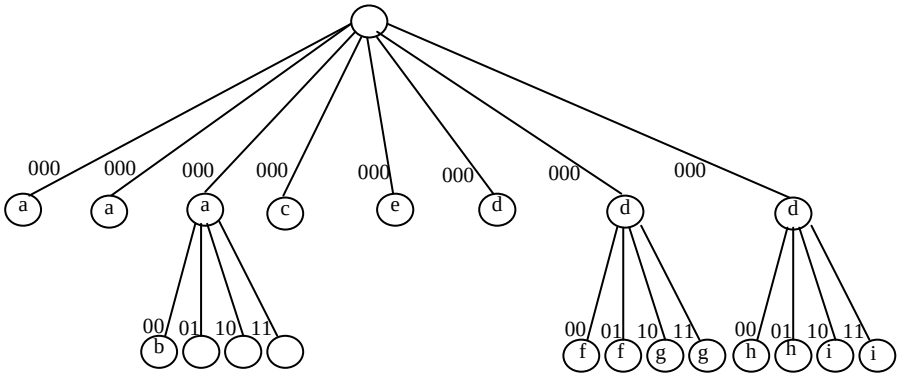


Figure.2.3. Trie multibit à préfixes disjoints

Dans la figure.2.4 nous pouvons voir le décodage d'une adresse IP pour trouver l'information de routage. Les premiers 24 bits de l'adresse IP sont utilisés comme un index dans la mémoire du première niveau (premier niveau du trie multi-bits). Si le premier bit de l'adresse entrée est 0, l'entrée contient l'information de routage, autrement l'information de routage doit être recherchée dans la mémoire du deuxième niveau du tri multi-bits.

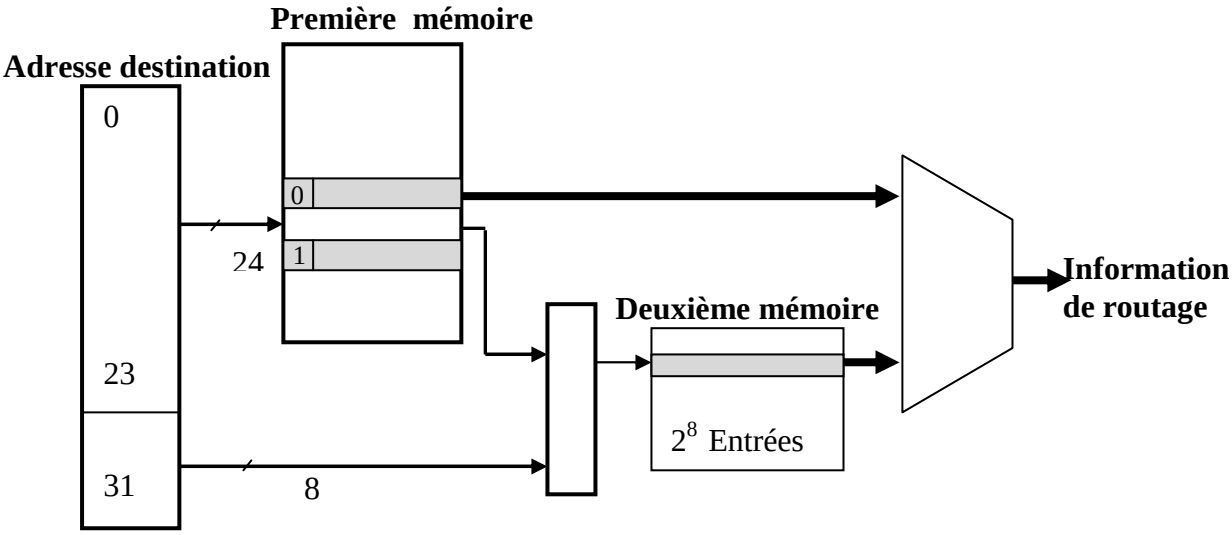


Figure.2.4. Schéma de Gupta (Implémentation matérielle de l'arbre multi-bit)

C. Trie multi-ary

> **Définition.2.1** [21] **:** Diverses transformations sont possibles sur les préfixes, mais l'une des techniques les plus courantes de transformation de préfixes est l'expansion des préfixes. L'expansion d'un préfixe signifie la transformation d'un préfixe en plusieurs préfixes plus long et plus spécifiques, mais qui couvrent tout le même intervalle d'adresses que le préfixe original. Ces nouveaux préfixes seront appelés préfixes dérivés depuis le préfixe original.
>
> Par exemple, l'intervalle d'adresses couvert par le préfixe 1* peut également être couvert par ces deux préfixes dérivés: 10*, 11*; ou encore, avec les quatre préfixes dérivés: 100*, 101*, 110*, 111*. En général, pour obtenir des préfixes dérivés de longueur *h* à partir d'un préfixe d'origine de longueur *l*, on à besoin d'ajouter au préfixe original toutes les chaines possible de longueur l-*h*. Alors, l'expansion d'un préfixe d'origine engendre 2^{h-l} préfixes dérivés de même longueur.

Un trie binaire inspecte un bit à la fois, et a potentiellement une profondeur de *W* pour des adresses de *W* bits. La profondeur maximale peut être diminuée *au W/K* en inspectant *K* bits à la fois. Ceci est réalisé en augmentant le degré de chaque nœud interne à 2^k. L'arbre généré est appelé *2^K-ary trie* et a un maximum de *W/K* niveaux. Le nombre de bits inspectés par l'algorithme de recherche du plus long préfixe à chaque nœud du trie désigné sous le nom du pas de parcours.

Des préfixes sont stockés dans un trie multi-ary de la façon suivante: Si la longueur d'un préfixe est un multiple intégral *de k* par exemple *mk*, le préfixe est stocké au niveau *m* du trie. Autrement, un préfixe d'une longueur qui n'est pas un multiple *de k* doit être augmenté vers plusieurs préfixes de longueur multiple d'un nombre entier k, par exemple un préfixe de longueur *K*-1 doit être expansé vers deux préfixes de longueurs *K*, et cela pour être stocké dans un 2^k -trie ary.

Exemple.2.1 : L'arbre 4-ary pour stocker les préfixes dans la table de routage du tableau.2.2 est représenté dans la figure.2.5. Tandis que les préfixes P2 et P3 sont stockés directement sans expansion, les longueurs des préfixes P1 et P4 ne sont pas des multiples de 2 et par conséquent ces préfixes doivent être expansés. P1 est expansé pour former les préfixes P11 et P12 alors que P4 est expansé pour former les préfixes P41 et P42.

Structure d'un nœud de l'arbre 4-ary

Next-hop (s'il existe un préfixe)			
ptr_{00}	ptr_{01}	ptr_{10}	ptr_{11}

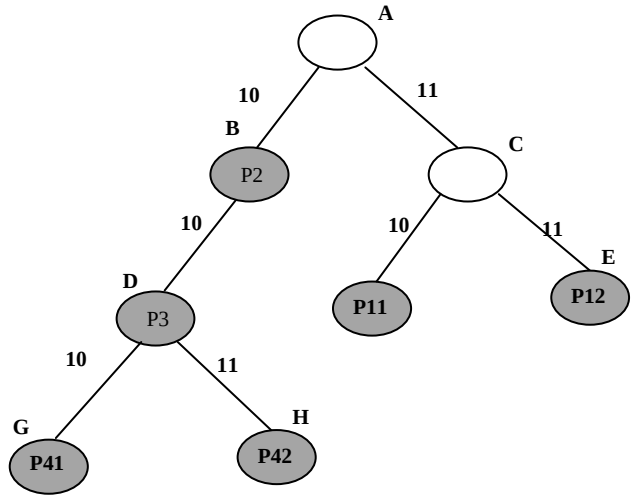


Figure.2.5. Un arbre 4-ary stocke les préfixes du tableau 2.2. Les nœuds gris stockant des pointeurs vers le Next-hop

L'expansion des préfixes augmente la consommation d'espace mémoire réservé pour le stockage de la structure de données arbre multi-ary a cause de:

- Le Next-hop correspondant à un préfixe doit être stocké dans des nœuds multiples de trie après expansion;
- Il y a un plus grand nombre d'indicateurs (nuls) inutilisés dans un nœud.

La diminution du temps de recherche du plus long préfixe est au détriment de coût d'espace mémoire. Le degré d'expansion commande cette compensation de stockage contre la vitesse dans la structure de données arbre multi-ary. Chaque nœud de l'arbre expansé est représenté par un tableau de pointeurs. Ce tableau a une taille de 2^k et le pointeur à l'index *j* du tableau représente la branche numérotée *j* et pointe au nœud fils à cette branche.

D. Arbres à chemins et à niveaux compressés (PC-Trie & LC-Trie)

Le trie [12], [22] est une structure de données utilisée pour stocker des chaines de caractères. L'idée est très simple: Chaque chaine est représentée par une feuille dans une structure arborescente et la valeur de la chaine correspond au chemin de la racine de l'arbre à la feuille.

A l'intérieur de cet arbre peut exister des nœuds avec un seul fils, alors ces bits doivent être parcourus même si aucune décision réelle de branchement est faite, le temps de recherche peut être plus long à cause du parcours non nécessaire dans certains cas. En outre, les nœuds avec un seul fils consomment un espace mémoire supplémentaire. Pour améliorer la performance temps et espace, une technique appelée compression de chemin peut être utilisée *(Path compressed Trie*) [18], [23], [24] et permet d'enlever ce problème et d'employer la compression de chemins, chaque nœud interne avec seulement un fils est supprimé. Alors on doit stocker le nombre de sauts dans chaque nœud qui indique combien de pas ont été sautés sur le chemin. La figure2.6.a schématise le stockage des préfixes du tableau.2.3 dans un arbre binaire, la version arbre à chemins compressés de l'arbre binaire de la figure.2.6.a est représentée par la figure.2.6.b.

Le nombre de nœuds dans l'arbre binaire à chemins compressés est exactement 2^n- 1 [23], où n est le nombre de feuilles dans l'arbre binaire.

La compression de niveaux [18], [24] est une technique utilisée pour compresser les parties denses de l'arbre binaire. Un LC-trie *(Level copressed Trie)* est créé à partir d'un arbre binaire comme suit : D'abord, la compression de chemins est appliquée au trie binaire. En second lieu, chaque nœud qui est enraciné à un sou-arbre complet de profondeur maximum est augmenté pour créer un nœud de degré 2^k [23]. Cette expansion est faite récursivement sur chaque sous-arbre de l'arbre binaire. L'idée est de remplacer les *i* niveaux complets les plus élevés de l'arbre binaire par un seul nœud du degré 2^i.

La version LC-trie de l'arbre binaire dans la figure.2.6.a à été représentée dans la figure.2.6.c.

N°	**Préfixe**	**N°**	**Préfixe**
0	0000	8	101001
1	0001	9	10101
2	0010	10	10110
3	010	11	10111
4	0110	12	110
5	111	13	11101000
6	100	14	11101001
7	101000		

Tableau.2.3. Exemple 2.3 de préfixes

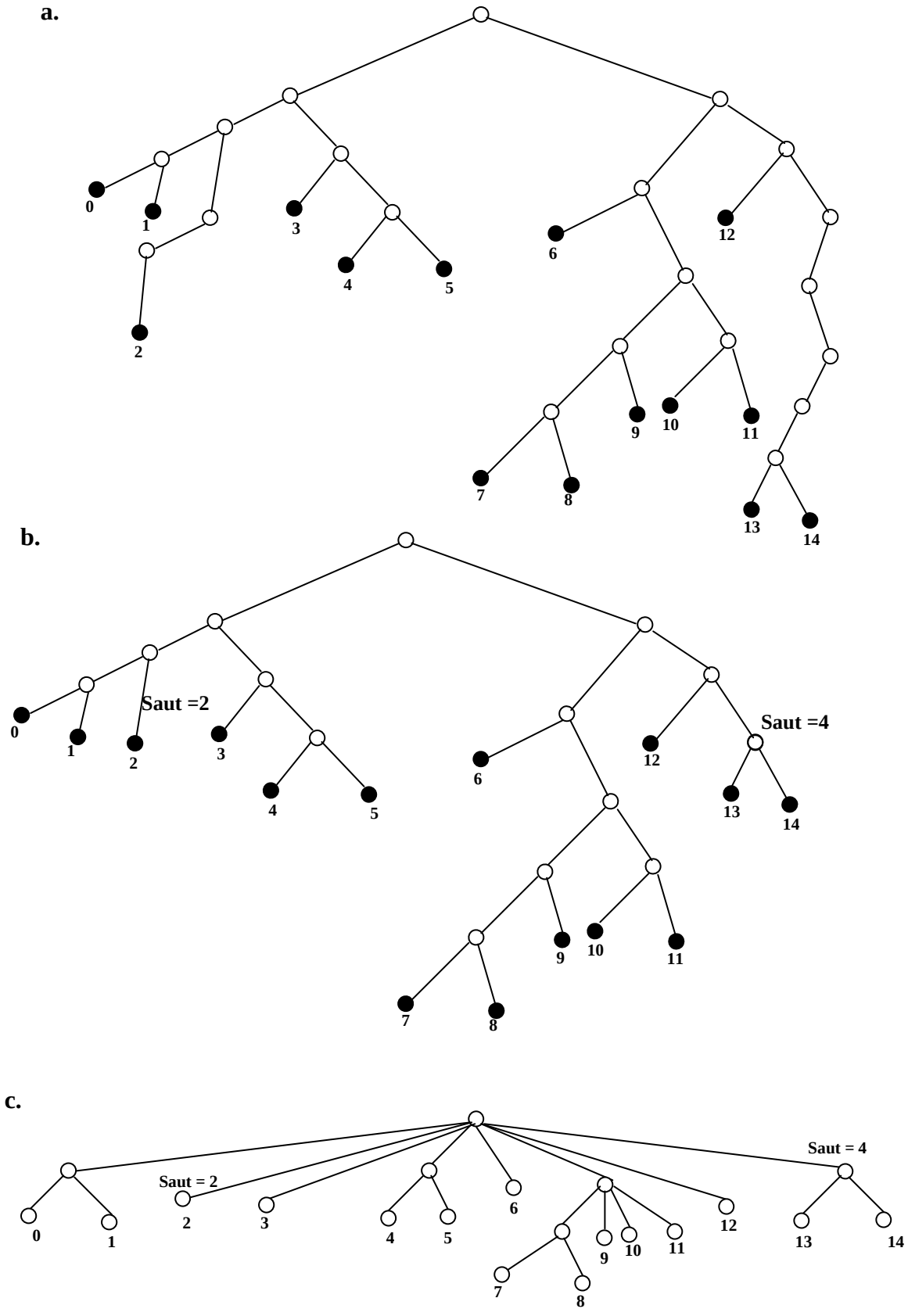


Figure.2.6. a. arbre binaire ; b. arbre à chemins compressés (PC-Trie) ; c. arbre à niveaux compressés (LC-Trie)

E. Le Trie PATRICIA

PATRICIA est une abbreviation pour "*Practical Algorithm To Retrieve Information Coded In Alphanumeric*". Un arbre PATRICIA [25], [26] est une variation d'une structure de données d'arbre binaire, avec la différence qu'il n'a aucun nœud de degré 1. Chaque chaîne est compressée dans un seul nœud dans l'arbre Patricia. Par conséquent, l'algorithme ne doit pas nécessairement inspecter toutes les adresses consécutives de l'adresse IP. Chaque nœud stocke alors un champ supplémentaire dénotant la position du bit dans l'adresse qui détermine la prochaine branche à prendre au niveau de ce nœud.

	Préfixe	Next-hop
P1	111*	H1
P2	10*	H2
P3	1010*	H3
P4	1101*	H4

Tableau.2.4. Exemple 2.4 de préfixes

Structure d'un nœud de l'arbre PATRICIA

Position du prochain bit	
Fils gauche	Fils droit

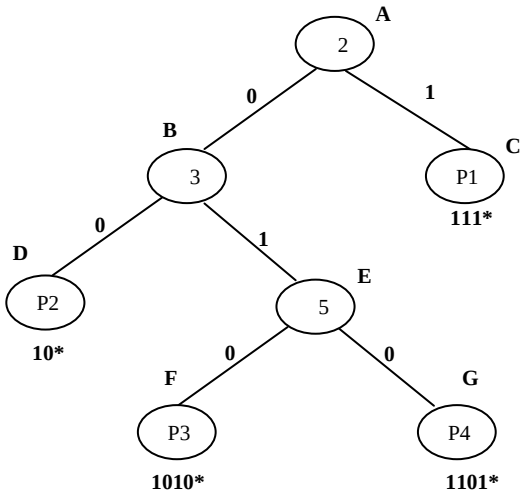


Figure.2.7. Le Trie PATRICIA pour l'exemple du tableau.2.4

La figure.2.7 représente l'arbre Patricia pour notre exemple de la table de routage. Etant donné qu'un arbre Patricia est un arbre binaire complet (c.-à-d., tous les nœuds sont de degrés 0 ou 2), alors, il a exactement N nœuds externes et N- 1 nœuds internes. Alors, la complexité de recherche dans l'arbre Patricia est O(N) [25].

Les préfixes sont stockés dans les feuilles de l'arbre Patricia. L'algorithme de recherche (*lookup)* descend l'arbre du nœud racine à un nœud feuille comme celui dans un arbre binaire. À chaque nœud, il sonde l'adresse pour le bit indiqué par le champ « position du prochaine branche » dans le nœud courant. La valeur de ce bit détermine la branche à suivre.

Quand l'algorithme atteint une feuille, il compare l'adresse IP avec le préfixe stocké dans la feuille. Ce préfixe est la réponse désirée si l'adresse IP correspond au préfixe. Autrement, l'algorithme doit périodiquement faire retour arrière et continue la recherche dans l'autre branche du nœud parent de cette feuille. Par conséquent, la complexité de recherche dans un arbre Patricia peut atteindre dans le pire des cas $O(W^2)$ [25].

F. L'arbre de priorité

Les auteurs de l'article [27], proposent un nouvel algorithme de recherche du plus long préfixe basé sur une structure de données appelée arbre de priorité (*priority trie*). La plupart des structures de données arborescente comportent de nombreux nœuds internes vides c-à-d ne stockent aucune information, ces nœuds vides consomme un plus d'espace mémoire et augmente le nombre d'accès a la mémoire dans le cas de la recherche sur l'arbre. En outre, les algorithmes de recherche comparer une adresse IP entrée avec les plus courts préfixes d'abord, et, poursuivre la recherche jusqu'à ce qu'une feuille est visitées, même si une correspondance est trouvée. L'algorithme proposé dans [27] utilise les nœuds internes vide pour stocker le plus long préfixe parmi les préfixes appartenait au sous-arbre de chaque nœud vide comme une racine. D'où l'espace mémoire occupé par l'arbre est réduit. Le nœud vide qui devient non vide est appelé nœud de priorité (*priority node*). La recherche par cet algorithme est immédiatement terminée sans rechercher l'arbre complet si une entrée donnée correspond à un nœud de priorité, et donc la performance de recherche est nettement améliorée.

La figure.2.9 représente l'arbre de priorité équivalent à l'arbre binaire classique de la figure.2.8. Les nœuds noirs représentent les préfixes situés dans leurs propres niveaux, et les nœuds grès représentent les préfixes de priorité. Dans la figure.2.9, le préfixe P3 est le plus long préfixe appartenait à l'arbre binaire enraciné par la racine vide, alors P3 doit être transféré au nœud racine. Le préfixe P1 est le plus long préfixe dans le sous-arbre enraciné par le nœud vide fils gauche de la racine, alors P1 doit aussi être transféré à la racine de ce sous-arbre, et ainsi de suite.

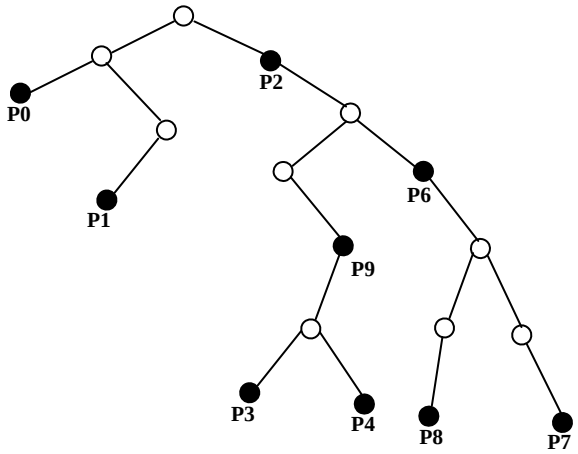


Figure.2.8. Arbre binaire classique

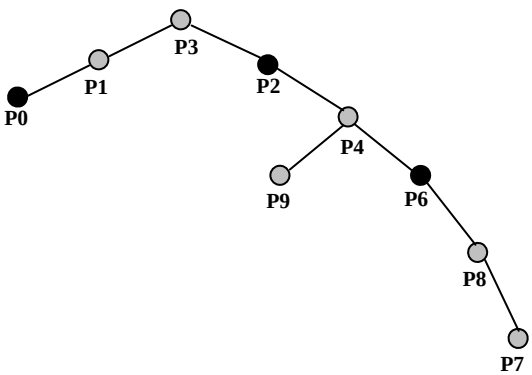


Figure.2.9. Arbre de priorité

2.2.2.2. Algorithmes de recherche binaire basés sur les points d'extrémités des plages d'adresses IP (Ranges search)

A. Transformation des préfixes à une plage d'adresses IP

Dans ce qui suit nous donnons une définition du problème de conversion d'un préfixe à une plage d'adresses. Nous utilisons une petite table de routage 'artificielle' tableau.2.5 comme un exemple.

> **Définition.2.2.** [28], [29], Un préfixe P représente un ensemble d'adresses dans une plage. Quand une adresse est exprimée comme un entier, le préfixe P peut être représenté comme une suite d'entiers consécutifs, exprimée par [*b*, *e* [, où *b* et *e* sont des entiers et [*b*, *e* [= {$x : b \leq x < e$, x est un entier}, [*b*, *e* [est définie comme une plage d'adresses pour le préfixe P, *b* et *e* sont définis comme l'extrémité gauche et l'extrémité droite du préfixe P, respectivement.

Préfixe	Next-hop	Préfixe	Next-hop
0*	A	10*	C
00010*	B	10001*	A
001*	C	1001*	A
01*	C	1011*	D
011000*	D	11*	D
0111*	A	110*	A
1*	B	1101*	B

Tableau.2.5. Exemple 2.5 de préfixes

> **Exemple.2.2.** Par exemple, pour les adresses de 6 bits, le préfixe 001* représente la plage d'adresses entre 001000 et 001111 (Sous forme décimale, entre 8 et 15 inclusivement). [8, 16 [est la plage du préfixe 001*. 8 et 16 sont l'extrémité gauche et l'extrémité droite du préfixe 001*, respectivement.

Deux préfixes distincts peuvent partager au plus une extrémité. Par exemple, dans le Tableau.2.5, le préfixe 001* à 8 et 16 comme points d'extrémités et le préfixe 01* à 16 et 32 comme points d'extrémités. Ils partagent le même point 16. Comme chaque préfixe peut être converti en deux points d'extrémités. Alors N préfixes peuvent être convertis en plus à 2N points d'extrémités différents. La figure.2.10 représente les plages et les points d'extrémités des préfixes dans le Tableau.2.5. Notez que 14 préfixes produire 17 points d'extrémités dans cet exemple.

Si deux plages consécutives ont le même *Next-hop* (information de routage), le point d'extrémité partagé peut être éliminé. Par conséquent, les deux plages peuvent être fusionnées en une seule plage. Par exemple, les plages [34, 36[et [36, 40[, ont le même *Next-hop* « A », nous pouvons les fusionner dans la plage [34; 40[. Nous pouvons utiliser le point d'extrémité gauche pour représenter la plage, par conséquent, affecter le *Next-hop* de la plage à l'extrémité gauche.

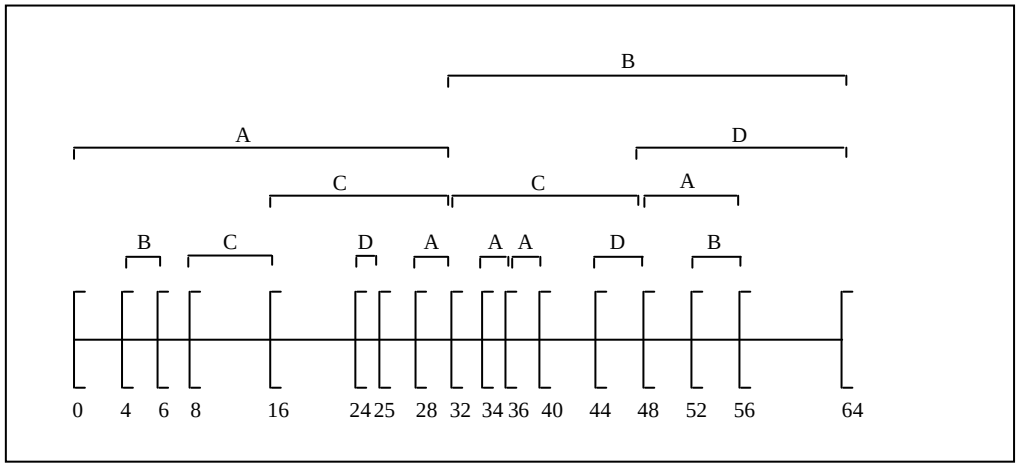


Figure.2.10. Les plages représentent les préfixes du tableau 2.5

Les auteurs des articles [28], [29], [30], [31] proposent des algorithmes pour convertir une table de routage en tuples sous forme (point d'extrémité, *Next-hop*).

B. Quelques exemples d'arbres de recherche avec plages d'adresses IP

Dans [32], les auteurs utilisent une recherche binaire sur les points d'extrémités représentant les préfixes, mais un prétraitement est nécessaire pour les opérations de mises à jour de sa structure de données. Dans [30], un nouvel algorithme est proposé pour améliorer les performances en utilisant une structure appelé B-arbre pour éviter les prétraitements de la structure de [32] mais chaque préfixe peut être stocké dans plusieurs nœuds. Dans [33], un nouvel algorithme est proposé pour améliorer les performances en stockant exactement un nœud pour chaque préfixe. La complexité de tous ces algorithmes peut atteindre dans le pire des cas *O(log N)* [30] tel que N est le nombre de préfixes dans la table de routage.

Les auteurs de [29] développent une structure de données qui utilisent les arbres de recherche avec priorités. Cette structure de données représente les plages d'adresses et permettre d'effectuer chacune des opérations insérer, supprimer et trouver la plage la plus spécifique avec une complexité *O(log n).*

Les auteurs de [28] développent un algorithme de recherche d'adresse IP qui utilise un arbre de plages d'adresses et non pas l'arbre des préfixes. Ils proposent à implémenter son algorithme dans une simple puce (*one single chip*). Son algorithme consiste à convertir le problème de recherche du plus long préfixe correspondant vers un problème de recherche dans des plages d'adresses IP [29], [30].

C. Structure de [31]

Dans [31] une autre structure arborescente est utilisée pour stocker les points d'extrémités des plages d'adresses IP (*height-balanced inorder-threaded binary search tree*).

Chaque préfixe est remplacé par ses deux points d'extrémités, et les points d'extrémités sont triés par ordre croissant. Le préfixe sera représenté par une valeur unique parmi les points

d'extrémités de la liste et chaque point d'extrémité dans la liste correspond à un préfixe unique, et chaque région entre deux points d'extrémités consécutives à un préfixe unique. Le préfixe correspondant à chaque région est précalculé et est associé à ses deux extrémités. En principe, trois préfixes sont associés à chaque point d'extrémité, à savoir L-BMP (*Less Best matching Prefix*), E-BMP (*Equal Best matching Prefix*), G-BMP (*Greater Best matching Prefix*). Exemple, soit l'exemple de la table de routage suivante :

Préfixe	**Next-hop**	**Préfixe**	**Next-hop**
001*	A	01101*	G
001010*	B	01100*	H
010100*	C	0100*	I
010101*	D	001*	J
0*	E	00010*	D
01*	F		

Tableau.2.6. Exemple 2.6 de préfixes [31]

Le tableau.2.7 représente la liste des points d'extrémités et leurs plus longs préfixes associés correspondent aux préfixes de la table de routage du Tableau.2.6.

N°	**Point d'extrémité**	**Valeur**	**L-BMP**	**E-BMP**	**G-BMP**
0	A_L	00 10 00 00	/	A	A
1	B_L	00 11 01 00	A	B	B
2	B_H	00 11 01 11	B	B	A
3	A_H	00 11 11 11	A	A	/
4	C_L	01 01 00 00	/	C	C
5	C_H	01 01 00 11	C	C	/
6	D_L	01 01 01 00	/	D	D
7	D_H	01 01 01 11	D	D	/
8	E_L	10 00 00 00	/	E	E
9	F_L	10 00 00 00	E	F	F
10	G_L	10 01 00 00	F	G	G
11	G_H	10 01 01 11	G	G	F
12	H_L	10 01 10 00	F	H	H
13	H_H	10 01 11 11	H	H	F
14	I_L	10 11 00 00	F	I	I
15	I_H	10 11 11 11	I	I	F
16	F_H	10 11 11 11	F	F	E
17	J_L	11 00 00 00	E	J	J
18	J_H	11 01 11 11	J	J	E
19	K_L	11 10 10 00	E	K	K
20	K_H	11 10 10 11	K	K	E
21	E_H	11 11 11 11	E	E	/

Tableau.2.7. Liste des points d'extrémités (en 8-bits) et leurs plus longs préfixes pour les préfixes de l'exemple du tableau.2.6 [31]

2.2.2.3. Algorithmes de recherche binaire basés sur les valeurs des préfixes

La plupart des méthodes proposées résoudront le problème de recherche du plus long préfixe en modifiant la structure de données arbre binaire classique (*trie*). La structure de trie divise l'espace de recherche en sous-espaces de recherche basée sur la valeur du caractère de la chaîne du préfixe à chaque étape. L'application directe de cette structure pour le problème de plus long préfixe correspondant ne donne pas de résultat très efficace. La figure.2.1 montre un exemple dans lequel la structure trie a été directement appliquée aux préfixes du tableau3.1. La structure de données divise l'espace de recherche en deux sous-espaces pour chaque bit de l'adresse IP, soit zéro ou un. Chaque préfixe est représenté par un chemin partant de la racine. Si un préfixe existe dans la table de routage, le nœud correspondant est marqué comme élément de données et identifié par un nœud noir dans la figure2.1, le reste (nœuds blancs) sont des nœuds vides, alors cette structure souffre d'insuffisances suivantes :

- Les nœuds blancs ne correspondant pas à n'importe quel élément de données dans la table de routage. Ils consomment un plus d'espace mémoire, augmente la hauteur de trie, et, par conséquent, prolonger le processus de recherche.
- Le temps de recherche correspond à la longueur des préfixes. Dans le pire des cas, ceci peut être 32 en IPv4 et 128 en IPv6.
- Un autre problème intrinsèque, les préfixes les plus court sont situés dans des niveaux plus élevés et donc ils sont comparés plus tôt que les plus long préfixes. Par conséquent, même si une correspondance est trouvée, la recherche doit être poursuivie jusqu'à ce qu'une feuille.

Les algorithmes de recherche binaire sur les valeurs de préfixe servent à proposer des trie sans nœuds vides c-à-d tous les nœuds de l'arbre stockent des données utiles. Afin d'effectuer la recherche sur les valeurs binaires des préfixes, les préfixes doivent être triés en fonction de leurs valeurs binaire.

Les mécanismes qui se basent sur les valeurs binaires des préfixes prévoient une série de nouvelles définitions [34] pour la comparaison des préfixes de longueurs différentes dans le tri.

A. Le tri des préfixes

La structure arbre binaire de recherche peut être appliquée à des nombres et des textes car ils peuvent être triés. En effet, le tri se comporte comme une fonction qui donne la position relative de chaque élément de données dans l'espace tri. A cet effet, l'objectif est de trouver une fonction de tri pour les préfixes qui prend n'importe quel préfixe et trouve sa position par rapport au reste.

> **Définition.2.3.** : Supposons qu'on à deux préfixes $A = a_1a_2\ldots a_n$, $B = b_1b_2\ldots b_m$, où ai et bi égal 0 ou 1 pour tous i et j :
>
> 1) Si n = m et les deux chaînes ont la même longueur, les valeurs numériques de A et B sont comparées.

2) Si n ≠ m (supposer que n < m), les deux sous-chaines ($a_1a_2…a_{n)}$ et ($b_1b_2…b_n$) sont comparées. Le préfixe avec la grande valeur numérique est considéré comme plus grand (plus petit) si les deux sous-chaînes ne sont pas égales. Si les deux chaînes ($a_1a_2…a_{n)}$ et ($b_1b_2…b_n$) sont égales, Alors, le $(n+1)^{ieme}$ bit de la chaîne B est considéré. Si b_{n+1} est 1 alors B > A sinon, B ≤ A

Exemple.2.3 : Prenons les quatre préfixes suivant : 1011, 1101, 11101 et 101101 comme un exemple. Il est claire que le préfixe 1101 est plus grand que le préfixe 1011 est plus petit que 11101. Le préfixe 1011 est supérieure à 101101 puisque les quatre premiers bits des deux préfixes sont les mêmes mais le 5^{ieme} bit du deuxième préfixe est zéro.

Définition.2.4. Supposons qu'il existe deux préfixes A = $a_1a_2…an$, B = $b_1b_2…b_m$. L'un de ces préfixes correspond à l'autre, si n = m et les deux chaînes sont identiques, ou bien, (Supposons que n < m), les deux sou-chaînes ($a_1a_2…a_n$) et ($b_1b_2…b_n$) sont les mêmes, sinon, A et B ne sont pas correspondants.

Définition.2.5. Deux préfixes A et B sont disjoints s'ils ne sont pas un préfixe ou une sous-chaîne d'autres préfixes

Définition.2.6. Un préfixe A est appelé une enclosure s'il existe au moins un élément dans l'ensemble de données tel que A est un préfixe de cet élément de données.

Supposons qu'il existe un ensemble de préfixes disjoints. Alors, Dans un arbre binaire construit sur la base de la définition.2.3 on peut bien trouver le plus long préfixe d'une adresse IP donnée [34]. Par exemple l'arbre de la figure.2.11 représente les préfixes du tableau.2.8

Préfixe	**Next-hop**	**Préfixe**	**Next-hop**
10*	A	001100*	G
01*	B	1011001*	C
110*	C	1011010*	B
1011*	B	0100110*	H
0001*	D	01001100*	I
01011*	A	10110011*	J
00010*	E	10110001*	K
01011001*	F		

[Tableau.2.8. Exemple 2.7 de préfixes [34]

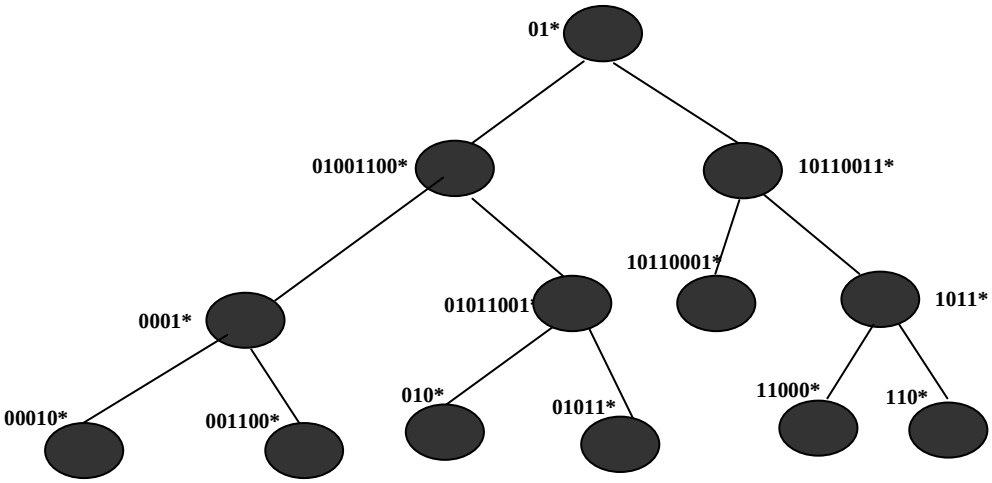


Figure.2.11. Arbre binaire équilibré pour l'exemple du tableau.2.8

La recherche binaire ne peut pas être appliquée directement sur la liste triée des préfixes. Supposant qu'un paquet entrant qui a le préfixe A en tant que meilleure correspondance, il est possible d'exister des préfixes qui ont le préfixe A comme sous-chaîne. Si l'on compare ces préfixes avant le préfixe A, la recherche binaire peut être adressée à la mauvaise moitié de la liste qui ne comprend pas le préfixe A. Par exemple, supposons que la liste triée est : 00*, 1*, 110101*, 1101*, et 111*, ainsi que l'adresse entrée est 110100. Etant donné aussi que le préfixe 110101* est au milieu de la liste triée, alors, l'entrée est comparée avec ce préfixe. L'entrée est plus petit que le préfixe 110101*, donc la recherche est orienter à gauche du préfixe 110101*. Par conséquent, la meilleure correspondance du préfixe de l'adresse d'entrée est le préfixe 1101*.

L'algorithme basé sur l'arbre BPT (*Binary Prefix Tree*) de [34] résout ce problème en comparent les préfixes enclosures avant les préfixes descendant. L'arbre BPT ne dispose pas de nœuds internes vides, et donc il a l'avantage de la taille mémoire nécessaire. Mais, la profondeur de l'arbre BPT est en fonction de la hiérarchie des préfixe, ce qui rend la profondeur pourrait devenir très longue.

Pour réduire la profondeur de l'arbre, l'arbre de préfixes pondérés (WPT) [35] considère le nombre de descendants dans le choix de la racine de chaque niveau. L'arbre WBPT construit par une courte profondeur et est plus équilibré que BPT [34].

En utilisant le fait que les préfixes disjoints construiraient un arbre parfaitement équilibré [34], la proposition de [36] construit plusieurs arbre équilibrés de préfixes (MBPT : *Multiple binary prefix tree*) seulement avec des préfixes disjoints. L'arbre de préfixes disjoints (DPT) de [37] est construit par les préfixes de feuilles, et donc c'est un arbre parfaitement équilibré pour l'ensemble étendu des préfixes générés par les préfixes des feuilles.

B. Arbre de recherche LPFST (*Longest Prefix First Search Tree*)

L'arbre LPFST [38] range les plus long préfixes dans les niveaux supérieurs de l'arbre de tel sorte que la procédure de recherche (*lookup*) des informations de routage pour une

adresse IP se termine dès que l'adresse de destination du paquet entrant correspond à un préfixe dans certains nœuds internes avant d'arriver à une feuille. Dans cet arbre la longueur du préfixe à chaque nœud est supérieure ou égale à la longueur du préfixe de ses fils

L'arbre LPFST à moins accès à la mémoire par rapport aux arbres binaire et l'arbre Patricia [25]. Le LPFST n'a aucun nœud vide, ce qui diminuer le nombre de nœuds de l'arbre.

Structure d'un nœud de l'arbre LPFST

Chaque nœud est soit de type 0 ou 1. Un nœud de type 0 ne comporte qu'une seule entrée de la table de routage, alors qu'un nœud de type 1 comprend deux entrées de la table de routage.

Type: 0 ou 1

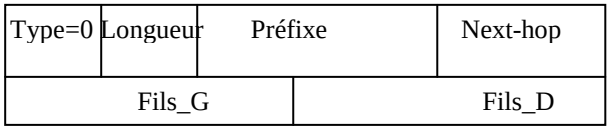
Longueur: La longueur du préfixe

Préfixe: Le préfixe

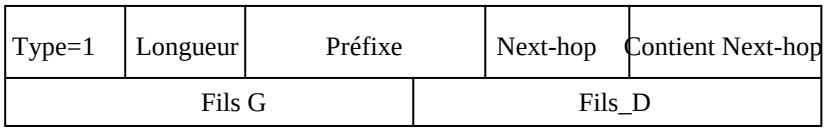
Next-hop: Le saut suivant (information de routage)

Contient next-hop: Le saut suivant du deuxième préfixe stocké dans un nœud de type 1

Fils_G/Fils_D: Pointeurs vers les fils gauche et droit du nœud ou bien NUL.



Un nœud de type 0



Un nœud de type 1

Figure.2.12. Structure d'un nœud de l'arbre LPFST [38]

Préfixe	Next-hop
00*	A
01*	B
010*	C
0110001*	D
0111*	E
10*	F
11000*	G
11001000*	H
11001001*	I
1101111*	J

Tableau.2.9. Exemple 2.8 de préfixes

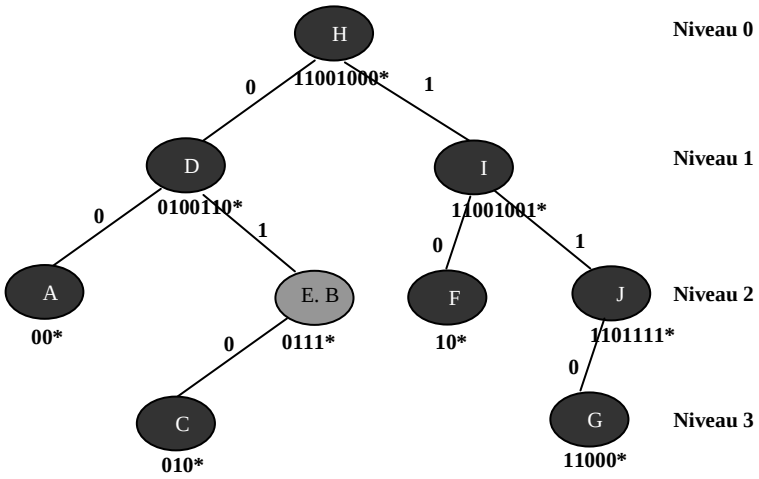


Figure.2.13. Le Trie LPFST pour l'exemple du tableau.2.9

Algorithme de recherche du plus long préfixe

Quand un paquet arrive au routeur, l'adresse de destination (noté DA) du paquet est extraite et l'algorithme de la figure.2.14 est exécuté. L'algorithme de recherche se termine immédiatement lorsque l'adresse IP correspond à un préfixe stocké dans un nœud de l'arbre, puisque aucun fils du nœud n'aurait un plus long préfixe qui peut correspond à l'adresse IP. Si le nœud est de type 1, le nœud doit contenir un préfixe dont la valeur correspond au chemin de la racine vers le nœud lui-même. Alors la procédure de recherche doit se poursuivre, car il n'est pas sûr que le préfixe contenu est le plus long.

```
Algorithme lookup (DA, y, Niveau)
{
Next-hop =  route par défaut
TantQue y ≠ Null Faire
{
    Si Get(DA, O, Ly ) ≡ Py Alors  Return Oy ;
    Si y est un noeud de type 1 Alors Next-hop = Contient Next-hop de y
   Si Get(DA, Niveaul, 1) ≡ 0 Alors y = Fils_G(y) ; Sinon y = Fils_D(y) ;
   Niveau++
}
Retutn Next-hop
```

Figure.2.14. Algorithme de recherche du plus long préfixe par l'arbre LPFST [38]

La fonction Get (X, i, k) renvoie la valeur de k-bits de l'adresse X commençant en position *i* et terminant en position *i + k -1*, où le bit de gauche de X est 0. Par exemple: Get (01011,2,3) = 011.

Exemple.2.4 : Considérons le LPFST dans la Figure.2.13, en supposant un paquet entrant avec l'adresse IP = "01100010". Tout d'abord, l'algorithme de recherche compare l'adresse IP avec le préfixe de la racine, et constate qu'elles ne le correspondent pas. L'algorithme se poursuit alors de comparer l'adresse IP avec le fils gauche de la racine (0110001*/ 7; D), et il correspond à l'adresse IP. Le processus de recherche se termine immédiatement et retourne le next-hop "D".

C. Arbre binaire équilibré avec vecteurs [39]

Les auteurs de [39] proposent un nouvel algorithme de recherche binaire équilibré basé sur l'arbre de recherche binaire. Cet algorithme construit un arbre de recherche binaire uniquement avec des feuilles de l'arbre binaire classique et met dans chaque nœud un vecteur des informations des préfixes imbriqués par rapport à ce préfixe. Notez que l'arbre binaire de recherche construit est parfaitement équilibré.

L'arbre de [39] est construit en deux étapes. Tout d'abord, un vecteur est construit pour chaque préfixe feuille de l'arbre binaire. Deuxièmement, les préfixes des feuilles avec un vecteur sont triés dans l'ordre croissant et stockés dans un tableau.

Définition.2.7 [39]**:** Dans la figure.2.15, soit P_l un préfixe feuille de trie binaire tel que P_0, P_1, P_3, P_4, P_7. Un vecteur du préfixe est composé de $n(P_l)$ éléments comme $V = [v_1, v_2, \cdots, v_{n(Pl)}]$. Si $S(P_l, k) = P_i$ pour $k = 1, \cdots, n(P_l)$, $v_k = Y_i$ où Y_i est le port de sortie, à lequel l'entrée est transmise une fois que le préfixe P_i est le meilleur préfixe correspondant. S'il n'ya pas de P_i tel que $S(P_l, k) = P_i$, v_k est un élément nul et représenté comme ø. Le vecteur du préfixe n'a pas besoin de maintenir la valeur du préfixe complet pour les préfixes imbriqués, mais seulement sur sa longueur. Le préfixe imbriqué est obtenu en considérant le nombre exacte de bits d'un préfixe feuille. Pour l'exemple de la Figure.2.15, les vecteurs des préfixes 110100* et 11111* sont $[Y_2, O, O, Y_5, ø, Y_3]$ et $[Y_2, ø, Y_6, ø, Y_7]$, respectivement.

Exemple.2.5 : Soit l'arbre binaire de la figure.2.15, L'arbre binaire avec vecteur de préfixes équivalent est représenté dans la figure.2.16.

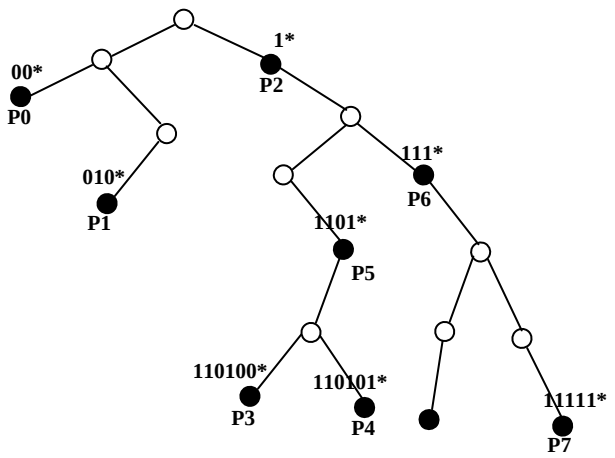


Figure.2.15. Exemple d'un arbre binaire classique

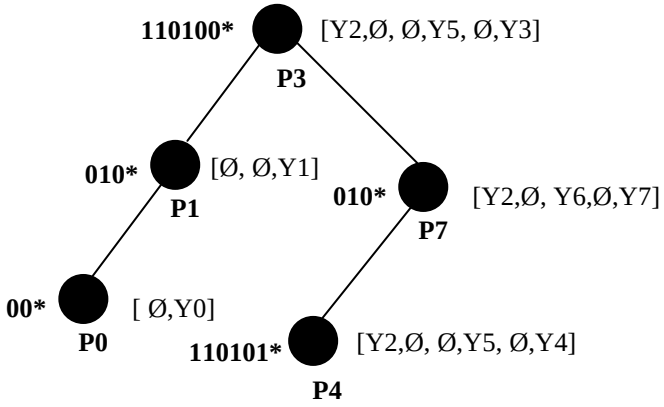


Figure.2.16. L'arbre binaire avec vecteur équivalent à l'arbre de la figure 2.15

Recherche du plus long préfixe dans l'arbre binaire avec vecteurs

Soit V(u) le vecteur préfixe d'un nœud u et P(u) est le préfixe feuille stocké dans le nœud u. Soit K la longueur du plus long préfixe correspondant et O représente le port de sortie correspondant au plus long préfixe. La recherche binaire est effectuée comme suit : Le nœud u est initialement définit comme le nœud racine, K est fixé à zéro, et O est réglé sur le port de sortie par défaut.

Si l'entrée A correspond au préfixe d'un nœud u, la recherche est immédiatement terminée et l'algorithme retourne ce préfixe. Si l'entrée ne correspond pas à ce préfixe, le vecteur du préfixe est examiné seulement pour les longueurs supérieures à la longueur du préfix actuel K. La plus grande longueur du préfixe correspondant à l'entrée dans le vecteur remplace la longueur actuelle, et le port de sortie correspondant remplace le port O actuel. Si l'entrée est plus petite que le préfixe actuel, la recherche passe au fils gauche du nœud. Sinon, la recherche passe au fils droit du nœud. Ce processus est répété jusqu'à ce qu'il n'y ait plus de nœuds à suivre, et le port de sortie courant est retourné.

Algorithme	**Complexité de lookup**	**Complexité de stockage**	**Complexité de mise à jour**
Arbre binaire	W	NW	W
Arbre PATRICIA [25]	W^2	N	W
Arbre à chemin compressé [12]	W	N	W
Arbre multi-aray [21]	W/k	2^kNW/k	-
Arbre à niveaux compressés (*LC-Trie*) [22]	W/k	2^kNW/k	-

Tableau.2.10. Comparaison des complexités des différents algorithmes.

2.3. Solutions matérielles

La motivation principale pour la mise en matérielle de la fonction de recherche des informations de routage est la nécessitée d'accélérer l'opération de routage dans les routeurs qui n'est pas généralement être obtenu par les mécanismes logiciels. Par exemple, presque tous les produits à grande vitesse des principaux fournisseurs de routeur effectuent des recherches de routes dans le matériel. L'avantage des solutions logicielles est que la mise en œuvre du logiciel est plus flexible, et peut être facilement adaptée en cas de modifications des protocoles.

2.3.1. Mémoire adressable par contenu CAMs et Ternary CAMs

Il existe deux formes de base de CAM: binaires et ternaires. Les CAMs binaire supportent le stockage et la recherche des bits binaires (0 ou 1). Mais les CAM ternaire stockent les bits de trois cas : zéro, un, ou *don't care* bit (0,1ou X).

Les CAMs

La mémoire adressable par contenu CAM, *(Content-Addressable Memory)* est un type de mémoire informatique spécial utilisé dans certaines applications de recherche à très haute vitesse. Elle est aussi connue sous le nom de mémoire associative (*associative memory*, *associative storage*, ou *associative array*).

Contrairement aux mémoires informatiques standard RAM (*random access memory*) pour lesquelles l'application utilisatrice fournisse une adresse mémoire et la RAM retourne la données stockée à cette adresse, une CAM est conçu de telle manière à ce que l'application utilisatrice fournisse un mot de données et la CAM cherche dans toute sa mémoire pour voir si ce mot y est existe ou non. Si le mot est trouvé, la CAM retourne une liste d'une ou plusieurs adresses où le mot a été trouvé (et dans certaines architectures, elle renvoie également le mot de donnée, ou d'autres morceaux de donnés associées). Donc, une CAM est l'équivalent matériel de ce que l'on appelle un tableau associatif en logiciel.

Parce qu'une CAM est conçue pour chercher dans toute sa mémoire en une seule opération, elle est plus rapide que la RAM, dans toutes les applications de recherche. Au contraire de la RAM, qui à des cellules de stockage simples, chaque bit de mémoire individuel dans une CAM complètement parallèle doit avoir son propre circuit de comparaison pour détecter une correspondance entre le bit stocké et le bit d'entrée. La circuiterie additionnelle augmente la taille physique de la puce CAM ce qui augmente le cout de fabrication. La circuiterie supplémentaire augmente également la puissance de dissipation puisque chaque circuit de comparaison est actif à chaque cycle d'horloge. En conséquence, une CAM n'est utilisée que dans les applications spécialisées où la vitesse de recherche ne peut pas être atteinte en utilisant une méthode moins coûteuse.

Les TCAMs

La CAM binaire est le type le plus simple de CAM qui est utilisé pour la recherche de données sous forme de 1 et 0. La CAM ternaire permette un troisième état de correspondance appelé "X" ou "quelconque" pour un ou plusieurs bits dans le mot de donnée stocké, permettant l'ajout de flexibilité dans la recherche. Par exemple, une CAM ternaire pourrait avoir un mot stocké de "10XX0" qui correspondra aux recherches des mots "10000", "10010", "10100", ou "10110". La flexibilité de recherche additionnelle vient avec un coût additionnel par rapport aux CAMs binaires parce-que la cellule de mémoire interne doit encoder les trois possibilités d'état au lieu des deux de la CAM binaire. Cet état additionnel est typiquement implémenté en ajoutant un bit de masque (bit "*care*" ou "*don't care*" (quelconque)) à chaque cellule mémoire.

2.3.1.1. Utilisation des CAMs et TCAMs dans les routeurs Internet

Les paramètres de routage qui déterminent la complexité de l'opération de recherche de routes sont la taille d'entrée, la taille du tableau, la vitesse de recherche, et la vitesse de mise à jour du tableau. Actuellement, les tailles des tables de routage sont environ de millions d'entrées, mais sont de plus en plus rapidement croissent.

Il existe de nombreux mécanismes logiciel pour la fonction de recherche d'information de routage (première section du chapitre), mais ne peuvent pas satisfaire les exigences des applications actuelles. Presque toutes les approches algorithmiques sont trop lentes. En revanche, La mémoire CAM utilise le matériel pour effectuer une recherche en un seul cycle, ce qui entraîne une complexité de temps constante O(1) .Ceci est accompli en ajoutant des circuits de comparaison à toutes les cellules de la mémoire associative. La recherche est massivement parallèle. Le point fort du CAM par rapport aux approches algorithmiques est leur vitesse de recherche qui est très rapide. Le goulot d'étranglement actuel est la consommation d'énergie à cause de la grande quantité de circuits de comparaison activés en parallèle. Réduire la consommation d'énergie du CAM est un objectif clé de recherches actuelle.

Les CAMs ternaires sont souvent utilisées dans les routeurs, où chaque adresse est composée de deux parties: l'adresse du réseau (*netid*), dont la taille peut varier en fonction de la configuration du sous-réseau, et l'adresse de la machine (*hotid*), qui occupe les bits restants. Le routage est fait en consultant une table de routage maintenue par le routeur qui contient les préfixes identifiant les réseaux de destination connu, et les informations nécessaire pour acheminer les datagrammes à ces destinations. Sans CAM, un routeur doit comparer les adresses de destination du paquet à acheminer avec chaque entrée de la table de routage, effectuant une recherche bit-à-bit. S'ils sont égaux, les informations d'acheminement correspondant sont utilisées pour réexpédier le paquet. En utilisant un CAM ternaire pour la table de routage rend la tache de recherche beaucoup plus efficace. Les adresses sont enregistrées en spécifiant quelles bits machine sont quelconques, et donc la recherche dans le

CAM renvoie immédiatement l'entrée de routage correcte; à la fois la comparaison des bits des préfixes et des bits de l'adresse sont effectués par le matériel CAM.

2.3.1.2. Architecture des CAMs / TCAMs

Exemple.2.6. Un exemple d'une table de routage simplifié est représenté dans le tableau.2.11. Les quatre entrées de la table sont des préfixes de 5-bit, avec le bit X, correspondant à la fois un 0 et un 1 dans cette position. L'entrée de la première ligne indique que toutes les adresses dans la plage de 10100-10111 sont transmis au port A. Par exemple, si le routeur reçoit un paquet avec comme adresse de destination 01101, cette d'adresse correspond à la fois la ligne 2 et la ligne 3 dans le tableau. La ligne 2 est sélectionnée car elle a les bits les plus précises, en indiquant qu'il est l'itinéraire le plus direct vers la destination. Cette manière de recherche est appelée le plus long préfixe correspondant.

N°	Préfixe	Port de sortie
1	101XX	A
2	0110X	B
3	011XX	C
4	10011	D

Tableau.2.11. Exemple 2.9 de préfixes

La figure.2.17 représente un schéma simplifié d'un CAM ternaire 4 x 5 bits avec une architecture basée sur le circuit NOR. Ce CAM contient la table de routage du tableau.2.11 Les cellules de base CAM sont arrangées en quatre mots horizontal, tous les cinq cellules de longueur contenant à la fois les bits de stockage et les circuits de comparaison. Les lignes de recherche sont tracées verticalement sur la figure et diffusent les données de recherche sur les cellules CAM.

Les lignes de correspondance (*matchlines*) placées horizontalement indiquent si les données de recherche correspondent au mot de la ligne ou non. Une ligne de correspondance activée indique une correspondance et une ligne de correspondance désactivée indique une non-correspondance, ceci est appelé un décalage (*mismatch*) dans la littérature CAM. Les lignes de correspondance sont entrées à un encodeur qui génère le préfixe correspondant.

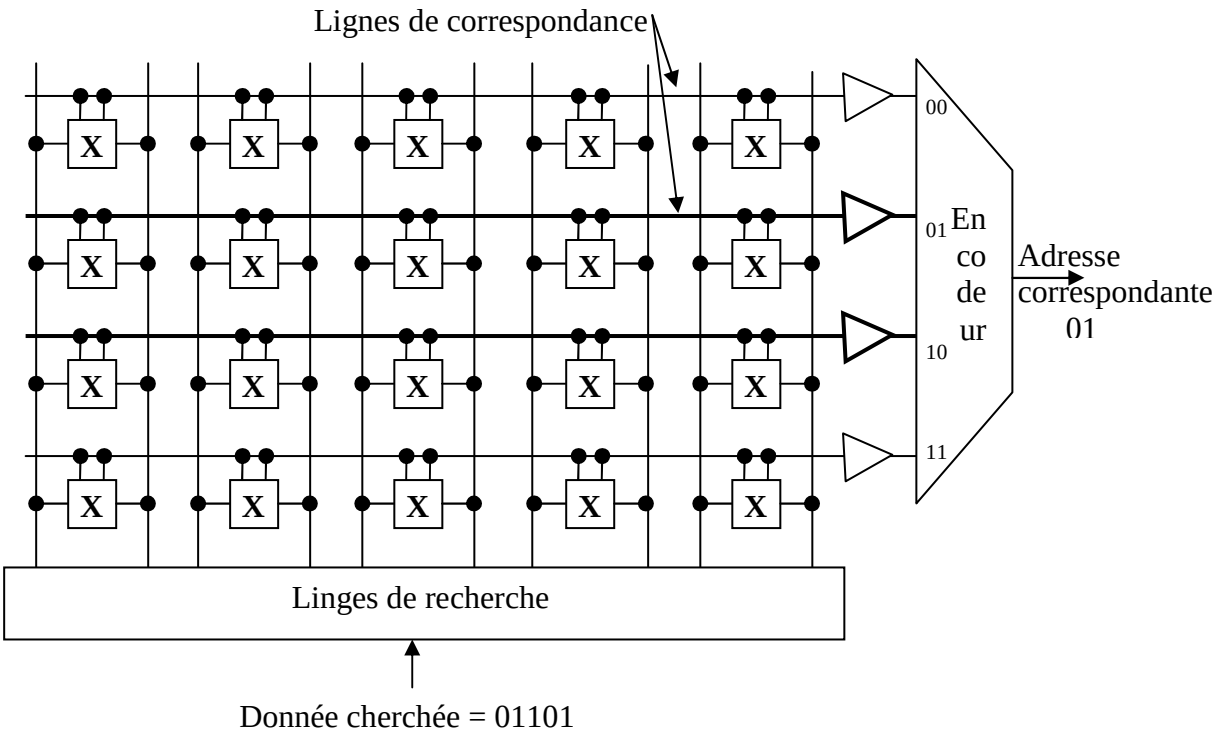


Figure.2.17. Architecture du TCAM [72]

2.3.1.3. Recherche dans les CAMs / TCAMs

Pour l'opération de recherche dans la CAM, d'abord mettant tous les *matchlines* temporairement à l'état de correspondance. Ensuite, les pilotes des lignes de recherches diffusent les données de recherche, 01101 dans la figure.2.17, sur les lignes de recherche. Ensuite, chaque cellule de base CAM compare son bit stocké avec les bits sur ses lignes de recherche correspondante. Les cellules de stockage d'un X fonctionnés comme si une correspondance est trouvée. Dans la figure, les deux lignes de correspondance du milieu restent activées, indiquant une correspondance, alors que les d'autres lignes de correspondance inactives, indiquant une non-correspondance. Et enfin, le codeur génère l'emplacement qui correspond l'adresse cherchée. Dans l'exemple, le codeur sélectionne la linge de correspondance avec la plus petite valeur numérique des deux lignes activées, générant l'adresse 01. Cette adresse est utilisée comme l'adresse d'entrée vers une mémoire RAM qui contient une liste des ports de sortie comme le montre la Figure.2.18.

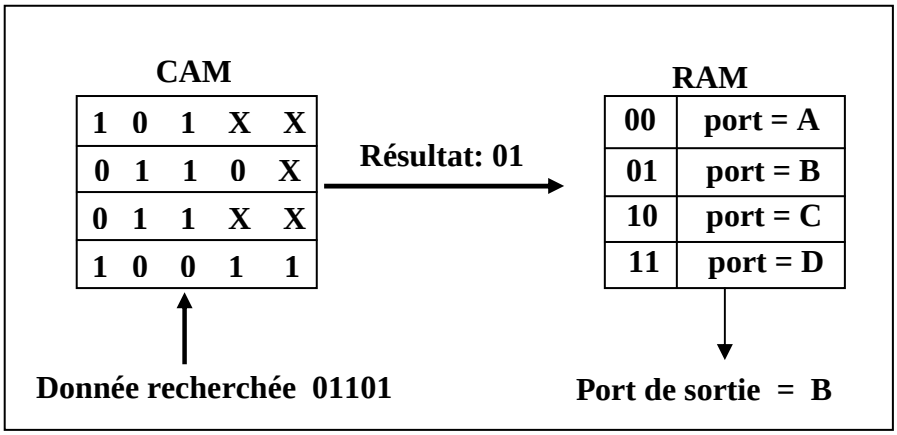


Figure.2.18. Architecture du CAM / RAM [72]

Ce système CAM / RAM est une implémentation complète d'un mécanisme de recherche du plus long préfixe correspondant pour une adresse IP. La sortie du CAM est utilisée comme un pointeur pour récupérer des données associées de la RAM. Dans ce cas, la donnée associée est le port de sortie (*Next hop*). La recherche dans le système CAM / RAM peut être considérée comme un dictionnaire de recherche où les données de recherche sont les bits de l'adresse IP (des mots) entrée pour être interrogée, et la RAM contient les définitions de mots (ports de sortie).

2.3.1.4. Architectures des CAMs / TCAMs proposées pour l'opération de lookup

Les approches matérielles utilisent généralement des circuits ou des schémas dédiés pour la recherche de l'information de routage [14], [40], [41]. Les techniques les plus populaires sont les mémoires à contenu adressable (TCAM) parce que leurs temps de recherche est de complexité O(1). Les TCAMs résolvent ce problème du plus long préfixe correspondant d'une façon plus rapide. Cependant, le principal inconvénient de l'utilisation de TCAMs dans les routeurs est que TCAM consomme une grande quantité d'énergie pour chaque opération de recherche puisque chaque cellule TCAM est activée pour chaque recherche. Il y a eu un nombre important de recherches pour tenter de réduire la consommation d'énergie dans les TCAMs [42] [53], [43], [44], [45], [46], [47] [48], [49], [50], [51], [52].

Panigrahy et Sharma [42] ont introduisent une architecture d'une TCAM paginée pour réduire la consommation d'énergie dans les routeurs. Leurs schéma partitionne l'ensemble des préfixes en huit groupes de tailles égales, chaque groupe se trouve sur une puce séparée TCAM. Une opération de recherche peut alors sélectionner et activer une seule des huit puces pour trouver une correspondance pour une adresse IP.

Lu et Sahni dans [53] proposent une technique qui utilise les SRAMs pour stocker des portions de préfixes, ainsi que leurs prochains sauts dans chaque mot du SRAM. Ce système réduit la taille du TCAM et minimise la consommation d'énergie de manière importante. Un inconvénient de ce système est que les algorithmes de mise à jour sont très complexes en raison de la nécessité de gérer les préfixes couvrant qui pourrait être reproduit plusieurs fois.

Wang et Tzeng [54] utilisent le système de feuilles poussées pour transformer les préfixes dans la table de routage en un ensemble de préfixes indépendants, qui sont ensuite stockées dans un TCAM (dans n'importe quel ordre).

Tania et Sartaj [55] proposent trois versions de l'architecture de double TCAM, avec des schémas de gestion de la mémoire pour l'exécution efficace et cohérente des mises à jour incrémentales. La première version de l'architecture est DUOS double TCAM avec SRAM simple, où les deux TCAM ont une simple SRAM associée et est utilisée pour stocker les sauts suivants. La deuxième version de l'architecture est DUOW dual TCAM avec SRAM large, où une ou les deux TCAMs ont des SRAMs associés qui sont utilisés pour stocker des suffixes ainsi que les sauts suivants. La troisième version est IDUOW double TCAM indexées avec des large SRAMs, dans laquelle une ou l'autre ou les deux TCAMs ont un TCAM comme index associé.

2.3.2. Mémoires cache

Des approches visant à améliorer les performances des routeurs par la mise en cache des informations de routage récemment recherchées sont aussi été proposées, tous les futurs paquets qui correspondent au même préfixe dans la table de routage trouveront leurs informations de routage dans la cache des préfixes et il ne sera donc pas nécessaire de se référer à la table de routage. La majorité de ces approches proposent des solutions matérielles c-à-d des processeurs avec mémoire cache des préfixes.

Dans les architectures classiques, une mémoire cache est placée entre le processeur et la mémoire de bas niveau contenant la table de routage IP. Durant la recherche, les adresses de destinations sont d'abord cherchées dans la cache. Si la recherche dans la cache est échouée, l'adresse est alors recherchée dans la mémoire de bas niveau qui est beaucoup plus lente que la cache. En conséquence, il est impératif que nous augmentions le nombre de recherches réussies dans la cache de sorte que moins de recherches doivent se référer à la mémoire de bas niveau.

Les caches traditionnels stockent les adresses IP [56], les préfixes [57], ou une combinaison des deux [58]. Au fil des ans, la mise en cache des adresses IP a été critiquée par les chercheurs à cause de l'augmentation exponentielle du nombre d'hôtes dans le réseau Internet, ce qui rend ce type de caches non efficaces. Ces dernières années, la mise en cache des préfixes a été bien abordée par la communauté des chercheurs. Parmi les raisons pour qu'elle soit efficace sont:

- Le nombre des préfixes qui est très peu par rapport au nombre d'adresses IP. La mise en cache des préfixes permettra de réduire la latence du réseau et d'améliorer les taux de recherche.
- Attribution des préfixes aux sous-réseaux basés sur la localisation géographique. Ainsi, toutes les hôtes d'une même zone géographique ont des préfixes similaires qui sont fréquemment référencé ce qui rend la mise en cache de ces préfixes utile.

Les premières recherches sur la mise en cache ont été stimulées par les travaux de [56]. Les auteurs proposent l'utilisation de la mise en cache des adresses IP pour les routeurs de vitesse de l'ordre de térabit. Ils développent un simulateur de cache d'adresses fréquemment référencés. Les auteurs montrent que le taux de réussite du cache est significatif (supérieure à 80%) pour une cache de taille raisonnable. Toutefois, la mise en cache des adresses IP démontré des taux de réussite inférieur par rapport à la mise en cache des préfixes.

Les auteurs de [59] développent une architecture indépendante des caches d'adresse IP connue sous le nom de cache d'adresses d'hôtes (HAC : *host address cache*). L'architecture visait à exploiter la cache du CPU. Les auteurs proposent à traiter les adresses IP comme des adresses de mémoire virtuelle et développent un algorithme de recherche basé sur deux structures de données: HAC et NART (*network address routing table*). Le HAC est utilisée au niveau-1 (L1) du cache dans le processeur. Pour l'opération de lookup, les adresses IP de 32 bits ont été considérées comme des adresses de mémoire virtuelle de 32-bit dans la cache L1.

Si la recherche dans la cache L1 n'a pas réussite alors la recherche est passée à la structure NART. Les auteurs ont traité la NART comme une cache de niveau 2. La NART se composait de trois tableaux contenant chacun les préfixes de la table de routage. Les préfixes ont été affectés à ces trois tableaux en utilisant une méthode complexe d'indexation et de marquage.

Le taux de réussite de la recherche dans la cache peut être maximisée par la mise en cache beaucoup d'entrées qui tenir autant d'informations récemment trouvées. Cependant, ceci peut ne pas être pratique. Néanmoins, il existe des techniques alternatives de "diviser le cache" qui sont proposées dans [60] et [57].

Les auteurs de [60] propose une cache des préfixe aligné (APC : *aligned-prefix caching*), divise la cache sur la base des longueurs de préfixe. APC crée deux caches alignées cache-24 qui met en cache tous les préfixes de longueur inférieur ou égal à 24 et cache-32 qui met en cache tous les préfixes de longueur comprise entre 24 et 32. En conséquence, APC montre des taux élevés d'accès au cache, pourtant, ce schéma est moins flexible ou dynamique. Cependant [61] propose une approche dynamique pour diviser la cache, les longueurs de préfixes sur les quelles la division est basé ne sont pas fixe mais varie en fonction du trafic.

L'idée pour augmenter le taux de réussite du cache de routage par compactage de table de routage (réduction) a d'abord été proposée dans [62]. Les auteurs ont démontré que les tables de routage peuvent être réduites par 48% de la taille. Exemple, soit les préfixes : P3 (01*, port2), P2 (011*, port2) et P1 (0110*, port3). Le P3 est un préfixe de P2 et ils ont le même port de sortie 'port2'. Ainsi P2 est redondant et peut être supprimé de la table de routage. La deuxième technique est l'extension du masque qui utilise l'outil EXPRESSO [63] pour la minimisation logique. Le minimiseur combine deux préfixes compatibles dans une seule entrée de routage. Par exemple, les deux préfixes 1100* et 1101* peuvent être combinés à 1X00 où X est un bit n'est pas défini (*don't care bit*). Les résultats de ces deux techniques augmentaient le taux de réussite du cache par 15%. Toutefois, le processus de minimisation logique à l'aide EXPRESSO est complexe et un peut lent.

Les auteurs de [58] développent une cache multizone pipeliné (MPC: *multizone pipelined cache*), une architecture à deux cache composée d'un TCAM et d'un CAM. La TCAM est utilisée pour mettre en cache les préfixes, et la CAM est utilisée pour mettre en cache les adresses IP. Dans cette architecture, la CAM est utilisée pour exploiter la localité temporelle et la TCAM est utilisée pour exploiter la localité spatiale.

Dans [64] les auteurs proposent de stocker les nœuds intermédiaires d'un arbre Patricia [25] dans une CAM. Les auteurs proposent d'utiliser trois CAMs pour stocker des préfixes de longueurs différentes. Plus précisément, les préfixes de longueur 8, 16 et 24 sont stockés dans ces trois CAMs. Quand un paquet arrive, les nœuds intermédiaires sont recherchés dans les trois caches en utilisant les 8, 16 ou 24 premiers bits du paquet. La correspondance de plus haute priorité est sélectionnée, et la recherche se poursuit ensuite en sautant au nœud intermédiaire (trouvé dans la cache) dans l'arbre de Patricia.

Les auteurs de [65] proposent un schéma de mise en cache des super-nœuds (*supernode*) pour réduire efficacement la vitesse de recherche dans les processeurs. Les super-nœuds sont des nœuds d'un arbre bitmap [66]. Un arbre bitmap est un sous-arbore compressé qui réduit le nombre de niveaux dans un arbre. Au court de l'opération de recherche, les super-nœuds récemment visités dans l'arbre bitmap sont stockés dans une mémoire cache à base de SRAM.

RRC (*Reverse Routing Cache*) a été proposée dans l'article [67]. RRC emploie une technique d'expansion minimale (ME : *minimal expansion*) afin de mis en cache les préfixes parent. Les préfixes parent sont des préfixes qui sont des préfixes pour d'autres préfixes. Par exemple, 172.19.24.0 / 8 est un préfixe de 172.19.24.0/24.

2.3.3. Architectures multiprocesseurs

Différentes approches matérielles sont proposées et déployées pour améliorer les performances des routeurs, dont la métrique de performance a été principalement le débit ou le nombre de paquets routés par seconde.

Les circuits (ASIC: *Application Specific Integrated Circuits*) [2] qui sont généralement utilisés pour implémenter les structures de données arborescentes (arbres binaires) utilisant une des mémoires à grande vitesse telles que les SRAMs. Les ASICs ne sont généralement pas préférés dans la conception des routeurs de haute gamme à cause de leur inflexibilité et le manque de programmabilité. Cependant les systèmes multiprocesseurs à usage général sont préférés pour les routeurs de haute gamme en raison de leur performance, de programmabilité et scalabilité .Le traitement des paquets de données dans ces routeurs est accompli grâce à des logiciels, d'où ils sont appelés routeurs IP logiciel (*software IP routers*).

Une architecture de base du routeur se compose de ports d'entrées, un processeur du routeur, un commutateur et des ports de sortie. La tendance actuelle dans les conceptions de routeurs est d'utiliser plusieurs unités de traitement des paquets au lieu d'une unique unité centralisée. Les architectures des routeurs de haute performance peuvent être divisées en trois organisations représentées dans la Figure.2.19. L'architecture de la figure.2.19.a relie les différents processeurs de traitement (*FE : Forwording Engine*) et les ports d'entrés via un bus partagé de haute vitesse. Les en-têtes des paquets entrants à partir d'un des ports de sortie doit être envoyé à un processeur pour effectuer la recherche des informations de routage. Le routeur multiprocesseur de [68] est basé sur cette architecture.

L'architecture parallèle de la figure.2.19.b remplace le bus partagé avec une matrice de commutation reliant tous les processeurs et les ports d'entrées. En outre, un processeur distinct est dédié pour l'exécution du protocole de routage, la comptabilité et autres tâches administratives qui ne sont pas sensibles au facteur de temps par rapport au traitement des paquets. Les processeurs de traitements sont les responsables de l'opération de recherche des informations de routage, de classification des paquets, et de mises à jour des en-têtes des paquets, qui sont des opérations sensibles au facteur temps. L'architecture des routeurs multigigabit [69] est conforme à la figure.2.19b.

La figure.2.19.c représente une architecture distribuée, où chaque port d'entrée a un processeur local et tous les traitements de routage, tels que l'opération de recherche des informations de routage sont exécutées localement dans les ports de sortie.

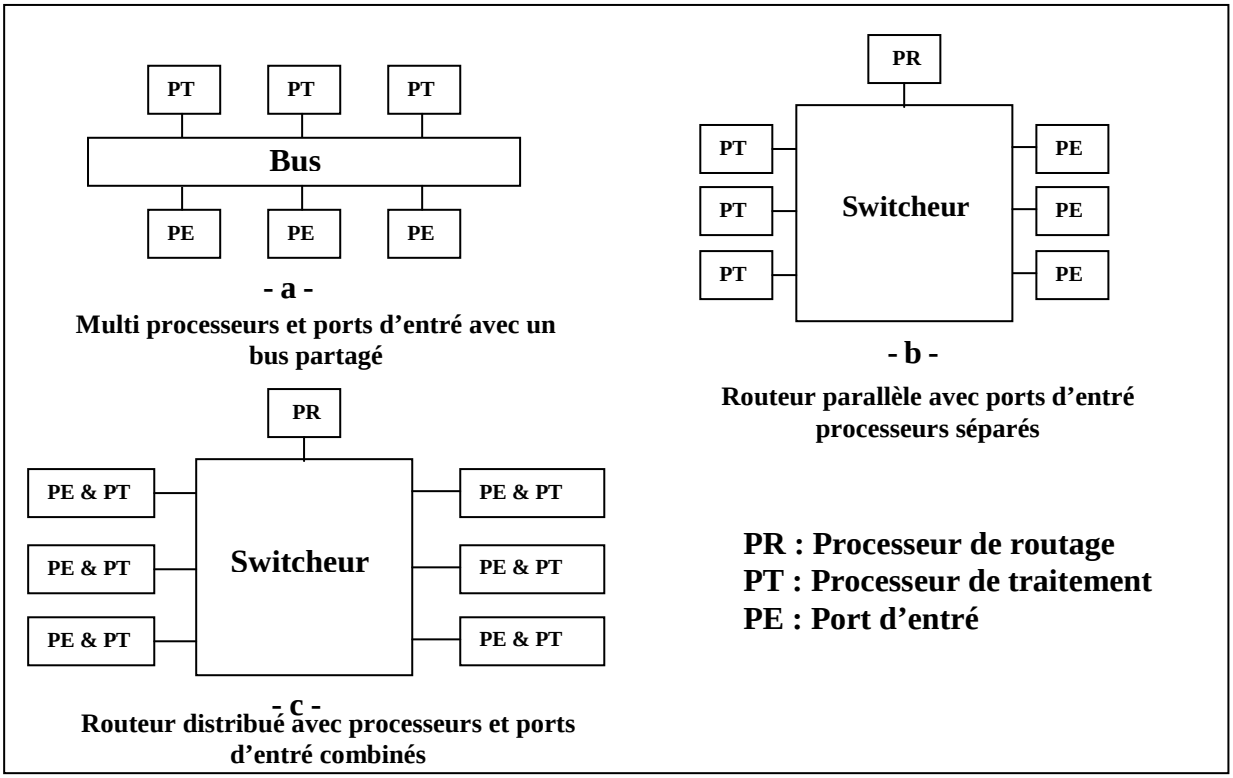


Figure.2.19. Architectures des routeurs

Il y a deux critères supplémentaires qui doivent être pris en considération lors de la conception d'un routeur multiprocesseur de haute performance [68]. La première est la programmabilité. Une architecture programmable avantageuse par rapport au circuit ASIC, car de nouveaux protocoles et fonctionnalités peuvent être ajoutés au routeur. Le second est l'utilisation de la mémoire. Les routeurs peuvent soit avoir toutes les structures de données répliquées pour chaque processeur de traitement, ou bien ont une structure de données partagée. Toutefois, comme la taille des structures de données (telles que la structure des arbres pour la table de routage) se développe et le nombre d'éléments de traitement augmente, les besoins en mémoire peuvent facilement dépasser les limites.

Les architectures des mémoires partagée peut être divisé en deux catégories [70], à savoir, Multi-Processeurs symétrique (SMP : *Symmetric Multi-Processeurs*) et mémoire cache-cohérent à accès non-uniforme (CC-NUMA : *Cache-Coherent Non-Uniform Memory Access*). Dans les architectures à mémoire partagée, un espace de mémoire globale est partagé par tous les processeurs du système. Chaque processeur possède une cache locale, qui peut stocker les blocs de données partagés récemment consulté.

2.3.3.1. Manipulation des tables de routage en parallèle

Pour améliorer les performances de la manipulation des tables de routage dans les routeurs, des architecture parallèle sont proposées [49], [71], toutes les architectures basées

sur le principe de partitionnement de la table de routage en plusieurs sous tables manipulées en parallèles pour chercher le plus long préfixe de chaque sous table et en suite sélectionner le plus long préfixe parmi les résultats des différentes sous tables. La différence essentielle entre ces propositions est résidée dans le critère de partitionnement de la table de routage originale et les principes de sélection du plus long préfixe parmi les résultats partiels obtenus. M. Akhbarizadeh et les autres [49] partitionnent la table de routage originale selon le port de sortie (Next Hop) de chaque préfixe c.-à-d. chaque sous table ne contient que des préfixes ayant le même port de sortie. Mais l'inconvénient majeur est que le nombre de sous tables est fixé et limité par le nombre de ports de sortie du routeur, et si le nombre de ports de sortie est petit, les tailles des sous tables ne sont pas loin de la taille de la table originale ce qui laisse le problème complexe.

D'autres solutions partitionnait la table originale en sous tables en basons sur la longueur des préfixes c-à-d chaque sous table ne contienne que des préfixes d'une telle longueur [71], alors le nombre des sous tables est fixé et égal au nombre de longueurs des différents préfixes (32 pour Ipv4 et 128 pour Ipv6) et en suite cherche le plus long préfixe à partir des résultats des sous tables par la recherche dichotomique par exemple.

2.4. Conclusion

Dans ce chapitre nous avons présenté une classification des différentes solutions proposées dans la littérature pour la recherche d'information de routage dans la table d'un routeur Internet, Plusieurs nouvelles approches ont été proposées pour accélérer cette opération; Il y à des solutions matérielles et des solutions logicielles.

La plupart des solutions logicielles proposées se basent sur les arbres binaires comme des structures de données pour le stockage du contenu de la table de routage, les structures arborescentes sont des simples structures de données et faciles à implémenter et manipuler, les chercheurs proposent en général des modifications sur l'arbre binaire classique de préfixes afin d'accélérer les opérations de recherche et de mises à jour de l'arbre des préfixes.

Les solutions matérielles (Hardware) pour le routage rapide des paquets IP sont classées en trois grandes catégories, à savoir, des solutions basées sur les mémoires associatives CAMs et TCAMs, sont conçues de telle manière que toutes les entrées soient comparées en parallèle; ces mémoires sont utilisées pour la recherche d'une donnée élémentaire (préfixe) en parallèle, Des solutions proposent d'utiliser le principe du cache, pour améliorer la performance de l'opération de recherche de route dans les routeurs internet par la mise en cache des informations de routage récemment recherchées. La majorité de ces approches proposent des solutions matérielles c.-à-d. des processeurs avec mémoire cache. Enfin, des architectures multiprocesseurs où la table de routage est manipulée en parallèle, toutes les solutions parallèles proposées sont basées sur le principe de partitionnement de la table de routage en plusieurs sous tables manipulées en parallèles.

Dans les chapitres suivants nous allons présenter nos solutions au problème de recherche d'information de routage dans les routeurs Internet.

Chapitre 3

Parallélisme et architectures parallèles

Avant de présenter notre architecture proposée, nous rappelons les fondements du parallélisme. Puis nous détaillions les deux classifications des architectures parallèles, celle de Flynn et de Ducon. En fin de ce chapitre, nous présentons les modèles de programmation parallèle et les étapes de développement d'une application exécutée sur une architecture parallèle.

3.1. Introduction

Le domaine que les problèmes de calcul concernent est de plus en plus large, et les systèmes sont de plus en plus compliqués. Le volume important de données cause des problèmes de stockage et de temps de calcul qui sont excessifs sur les machines séquentielles. La recherche de la puissance de calcul a donné naissance à différents types d'ordinateurs. En particulier, de nombreuses machines parallèles ont vu le jour depuis le début des années 80.

Notre travail est basé sur l'utilisation d'une architecture parallèle. Alors dans ce chapitre, nous présentons les fondements du parallélisme, les différentes classes des machines parallèles et les modèles de programmation parallèle.

3.2. Origine du parallélisme

Dans les ordinateurs à architecture traditionnelle dite de Von Neumann, toutes les opérations sont effectuées de manière séquentielle. Les instructions sont d'abord lues, puis décodées. Lorsque cela est nécessaire, les opérations sont également recherchées. L'opération est alors exécutée puis les résultats mémorisés. Aucune de ces opérations ne peut être lancée avant que l'opération précédente ne soit finie. Cette architecture a servi à la conception des premiers processeurs.

Les concepteurs se sont alors efforcés, tout au long des années, d'augmenter la puissance de calcul. Bien entendu, l'intégration de plus en plus de transistors sur une même surface permette des améliorations importantes, mais souvent jugées insuffisantes. Lorsque cela est insuffisant, l'augmentation de la puissance de calcul ne peut être obtenue que par une modification profonde de l'architecture. Celle-ci est guidée par un seul but : celui de paralléliser les actions.

Les besoins croissants des grandes applications scientifiques et militaires ont été le principal moteur économique du développement de ces nouvelles architectures. Les premières machines de calcul intensif, les machines vectorielles, sont apparues au milieu des années 70, notamment avec la création de *Cray Research* en 1974. Leur coût technologique au niveau du développement des architectures comme au niveau de leur utilisation, a freinée leur emploi d'une façon systématique. Aujourd'hui, un simple réseau d'ordinateurs individuels (stations de travail, PC) est considéré comme une machine parallèle. Le rapport coût/performance de ces machines parallèles est très intéressant. Cette évolution des performances résulte de la progression exponentielle de la vitesse des microprocesseurs, et de la densité d'intégration des microprocesseurs et des mémoires.

3.3. Sources de parallélisme

On peut avoir différentes sources de parallélisme :

Parallélisme de données: la même opération est effectuée par chaque processeur sur des données différentes.

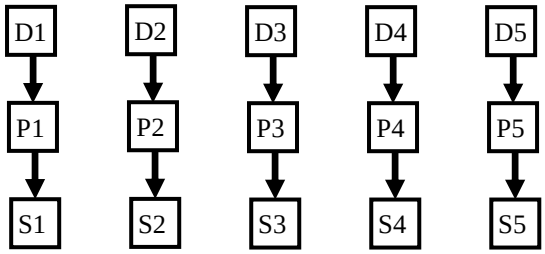


Figure.3.1. Parallélisme de données

Parallélisme de contrôle: des opérations sont réalisées simultanément sur plusieurs processeurs. Le programme présente des séquences d'opérations indépendantes qui peuvent être exécutées en parallèle

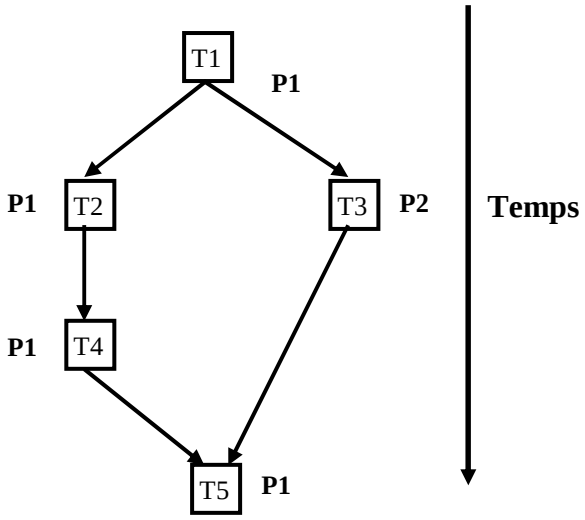


Figure.3.2. Parallélisme de contrôle

Parallélisme de flux: Les opérations sur un même flux de données peuvent être enchaînées (pipeline).

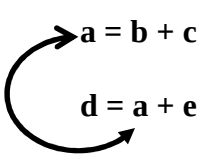
Temps	Tâche 1	Tâche 2	Tâche 3	Tâche 4	Tâche 5
T1	**Donnée 1**				
T2	**Donnée 2**	**Donnée 1**			
T3	**Donnée 3**	**Donnée 2**	**Donnée 1**		
T4	**Donnée 4**	**Donnée 3**	**Donnée 2**	**Donnée 1**	
T5	**Donnée 5**	**Donnée 4**	**Donnée 3**	**Donnée 2**	**Donnée 1**
T6	**Donnée 6**	**Donnée 5**	**Donnée 4**	**Donnée 3**	**Donnée 2**

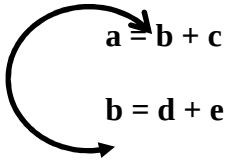
Tableau.3.1. Parallélisme de flux

3.4. Limites du parallélisme

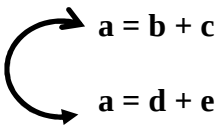
- **Les dépendances de données:**

Dépendance de flot





Dépendance de sortie



Alors les instructions doivent être indépendantes :

- Exécuter dans un ordre quelconque
- Simultanément

- **Les dépendances de contrôle:**

Soit les instructions suivantes :

I_1 : a = b + c;

I_2 : if (a < 0)

I_3 : {d = e + f;}

I_4 : g = d + h;

I_1 et I_4 sont à priori indépendantes mais selon la valeur de la variable 'a' I_3 peut être exécuté ou non ce qui entraînant une dépendance entre I_3 et I_4.

- **Les dépendances de ressources:**

Nombre insuffisant de processeurs pour effectuer les instructions en parallèle alors qu'elles sont indépendantes les unes des autres.

- **Rapport temps de communication / temps de calcul:**

Il n'est pas toujours avantageux de paralléliser une application. Les communications peuvent dans certain cas augmenter le temps d'exécution.

3.5. Architectures parallèles

Les architectures parallèles sont une des préoccupations actuelles du développement de l'informatique car elles offrent une solution pour l'amélioration de la vitesse des traitements. Une machine parallèle contient plusieurs unités de calcul de puissance variable opérant simultanément (parallélisme des calculs). Elle comporte un réseau d'interconnexion dont la nature, la structure, les méthodes de communication varient d'une machine à l'autre, afin d'échanger les résultats des unités de calcul (parallélisme des échanges). Elle utilise également des parallélismes d'exécution, ces trois types de parallélisme constituent les caractéristiques essentielles des machines parallèles.

L'animation de l'architecture utilisant tout ou partie de ces éléments nécessite des mécanismes de contrôle et de synchronisation pour les calculs, les échanges et les accès mémoires, qui se traduisent par des contraintes sur l'utilisation de ressources parallèle de la machine et définissent son comportement lors de l'exécution d'une application. Les architectures parallèles peuvent se distinguer par les processeurs (machines homogènes ou hétérogène), la topologie, la mémoire qui peut être partagée ou distribuée et la nature du réseau (liaison statique ou dynamique entre les processeurs). Alors les combinaisons introduisent un foisonnement de machines parallèles, rend difficile tous classement.

3.5.1. Classifications des architectures parallèles

La classification la plus simple et la plus employée est celle de Flynn [73]. Cette classification décrit les calculateurs selon les flots d'instructions et les flots de données, une architecture peut décoder une seule instruction ou plusieurs à la fois et peut manipuler une données ou plusieurs données à la fois. On obtient ainsi quatre catégories: SISD, SIMD, MISD et MIMD.

Cependant les progrès technologiques ont permis l'apparition de machines fonctionnant avec une logique totalement différente du processus séquentiel utilisé dans les machines de type Von Neumann : il s'agit des machines dataflow [74] où la nécessité de définir explicitement la synchronisation disparait. La classification de Flynn devient insuffisante et Duncan propose sa propre classification et différencie les architectures synchrones, les architectures asynchrones et les architectures mixtes [75].

Il existe encore deux autres critères de classification : l'organisation de la mémoire et la nature du réseau d'interconnexion entre les processeurs. La mémoire et le réseau d'interconnexion conditionnent en effet le parallélisme et le mode d'échange utilisé, les performances du réseau dépendant en grande partie de la structure utilisée.

3.5.1.1. Classification de Flynn

La classification de Flynn [73] repose sur la connaissance des flots d'instructions et des flots de données qui composent le calculateur. Elle distingue donc quatre types d'architectures, SISD, MISD, SIMD et MIMD

3.5.1.1.1. Architecture SISD (Single Instruction stream, Single Data stream)

Les architectures à flots d'instructions et de données uniques sont en général les processeurs séquentiels d'architecture de Von Neumann classique.

Les processeurs du commerce, de jeu d'instructions x86 ou les processeurs RISC (*Reduced Instruction Set Computer*) comme l'Alpha 21164, le powerPC, le MIPS ou le SPARK sont des exemples de cette catégorie d'architectures.

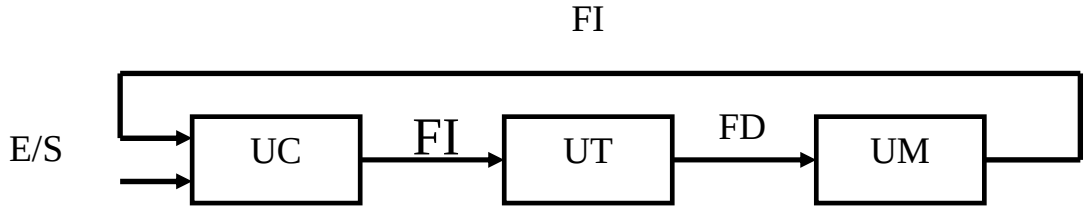


Figure.3.3. Architecture SISD.

L'unité de contrôle (UC), recevant son flot d'instructions (FI) de l'unité mémoire (UM), envoie les instructions à l'unité de traitement (UT), qui effectue ses opérations sur le flot de données (FD) provenant de l'unité mémoire.

3.5.1.1.2. Architecture SIMD (Single Instruction stream, Multiple Data stream)

Dans cette catégorie d'architectures, tous les processeurs exécutent la même instruction ou le même programme SPMD (Single Program Multiple Data) stocké en mémoire locale et issue d'une seule Unité de Contrôle sur des données différentes. Les processeurs fonctionnent de manière synchrone ; les instructions peuvent être simples ou complexes ; seul un sous-ensemble des processeurs peut avoir à exécuter une instruction.

Ce modèle est efficace surtout si on a un flot régulier sur des données régulières (auxquelles on accède de manière prévisible).

Une architecture parallèle SIMD est composée de trois parties essentielles :

- Les processeurs,
- Elément simple répétés un grand nombre de fois les interconnexions entre éléments,
- La partie contrôle du réseau et des processeurs.

Cette dernière peut être constituée soit d'une simple machine d'état, soit d'un processeur séquentiel évolué qui peut alors être un processeur CISC (**C**omplex **I**nstruction **S**et **C**omputer) ou RISC (**R**educed **I**nstruction **S**et **C**omputer).

Ce type d'architecture fonctionne sous le modèle présenté dans la figure suivante.

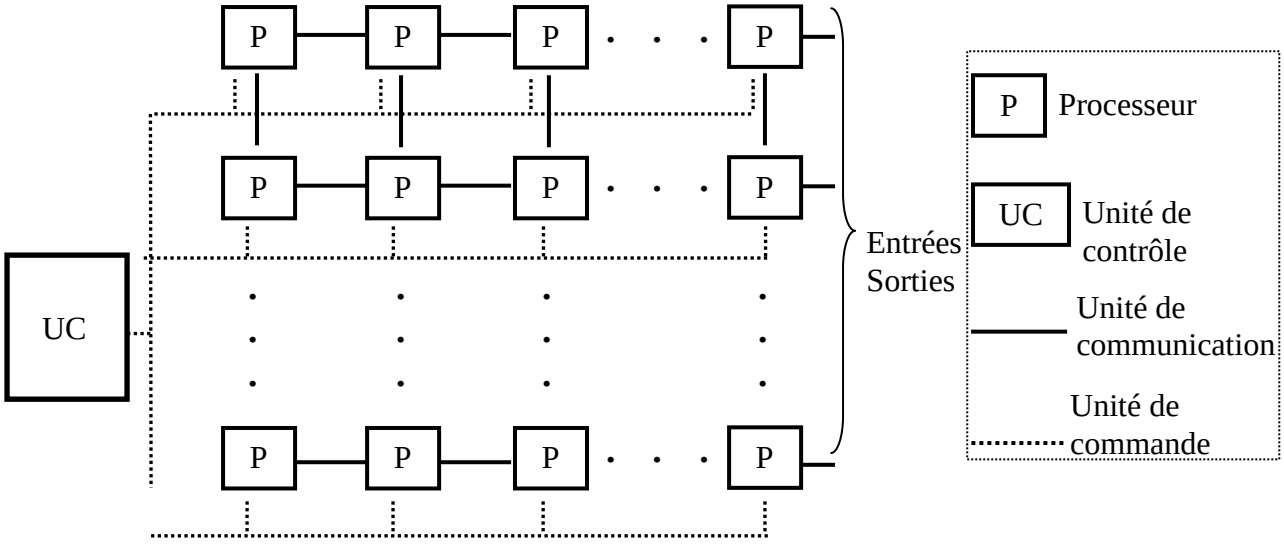


Figure.3.4. Modèle de fonctionnement des architectures SIMD

Les processeurs de ces machines n'ont pas de séquenceur et ont une puissance de calcul très faible par rapport aux machines séquentielles classiques. Ces processeurs sont constitués d'unités arithmétiques et logiques programmables. Chaque processeur possède une mémoire de faible capacité et exécute les instructions une à une, sous l'impulsion de l'unique contrôleur ou séquenceur. La puissance de calcul de ce type de machine est obtenue grâce au grand nombre de processeurs.

Souvent les processeurs doivent communiquer pour s'échanger des résultats intermédiaires, ce qui nécessite une coordination des activités. Ceci se fait de deux manières : soit l'utilisation d'une mémoire partagée (MP), soit l'utilisation de réseau d'interconnexion.

Mémoire partagée (MP)

C'est le modèle PRAM (*Parallel Random Access Machine*). Les *n* processeurs utilisent une mémoire partagée pour échanger des données (une étape élémentaire de complexité).

Au cours du temps les *n* processeurs vont accéder en lecture et / ou écriture en mémoire partagée, ce qui pose un problème de concurrence des accès.

Le modèle à mémoire partagée est théoriquement très puissant, mais peu réaliste ! En effet, on ne prend pas en compte la complexité globale du circuit d'accès à la mémoire qui dépend du nombre *n* de processeurs partageant cette mémoire et de la taille *m* de la mémoire. Problème si *n* et *m* sont grands ! Traditionnellement, on a $n < 64$ et *m* de l'ordre du Go.

Une solution pour diminuer le coût consiste à diviser la mémoire en *r* blocs de taille *m/r* chacun. La concurrence se fait uniquement au niveau des blocs…

Réseau d'interconnexion

Chaque processeur dispose d'une mémoire locale. Chaque paire de processeurs est connecté par un lien bidirectionnel et à chaque étape du calcul, un processeur quelconque P_i peut recevoir une donnée de P_j et envoyer une donnée vers P_k.

La topologie du réseau d'interconnexion (chaîne, grille 2D, arbre binaire, cube…) détermine la complexité des communications.

La Connexion Machine (CM) de TMC [76], CM-2 [77] [78] et MPP [77] sont des exemples de cette classe d'architectures. La connexion machine est constituée de 65536 processeurs élémentaires 1 bit. Un mécanisme de masquage permet d'exécuter l'instruction courante sur certains processeurs, alors dénommés actifs par rapport aux autres processeurs qui attendent la fin de l'instruction pour passer de l'état passif à celui d'actif. Le réseau d'interconnexion est un hyper cube.

Les architectures SIMD ont été plus ou moins abandonnées au début des années 90 en raison de leur manque de souplesse, sauf pour certaines applications comme le traitement d'images et le traitement de signal. Néanmoins, le fait d'avoir des flots séparés pour traiter les données reste une solution efficace pour le calcul rapide sur un grand nombre de données.

3.5.1.1.3. Architecture MISD (Multiple Instruction stream, Single Data stream)

Il y a plusieurs flots d'instructions agissant sur un seul flot de données. On a n > 1 processeurs partageant une unique mémoire.

A chaque étape de manière synchrone, une donnée est traitée parallèlement par les n processeurs qui exécutent chacun une instruction spécifique.

Cette catégorie regroupe les machines spécialisées de type « systolique », dont les processeurs, arrangés selon une topologie fixe. La machine systolique permette de résoudre les problèmes de baisse de performance dus aux limitations des entrées/sorties de toute structure. Ces calculateurs sont toutefois difficiles à mettre en œuvre et manquent de souplesse. Par conséquent, il en existe assez peu, et ils sont souvent dédiés à des applications particulières.

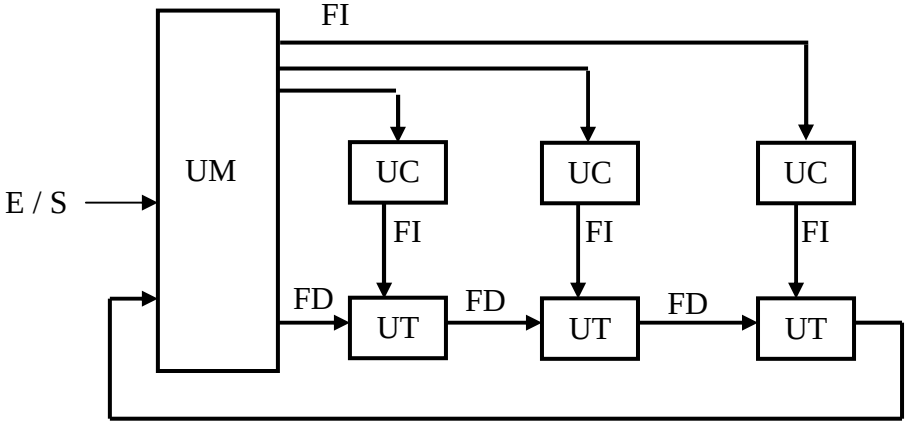


Figure.3.5. Architecture MISD.

3.5.1.1.4. Architecture MIMD (Multiple Instructions stream, Multiple Data stream).

Des flots d'instructions différentes qui s'exécutent sur des données différentes. Un ensemble de processeurs dont chacun d'entre eux à sa propre mémoire contenant des données locales sur lesquelles s'exécute son propre programme. C'est le modèle de la plupart des machines parallèles.

En général les processeurs de ces machines sont très nombreux, mais ils ne sont pas puissants. Les machines MIMD fonctionnent en mode SPMD (Single Programme Multiple Data) lorsqu'un même programme s'exécute sur tous les processeurs.

Cette forme de parallélisme est celle qui à été le plus largement étudiée et implémentée. Par exemple, le premier Cray-1 et les machines massivement parallèles à usage général comme le CrayT3D et T3E de SGI et le Ncube [79]. Tous les supercalculateurs actuels utilisent le modèle MIMD plus au moins respectueux de la définition générale.

On distingue habituellement deux sous-classes de MIMD [80], selon que les processeurs de la machine ont accès à une mémoire commune, on parle alors de MIMD à mémoire partagée, «multiprocessor » figure 3.6, ou disposent chacun d'une mémoire propre MIMD à mémoire distribuée [81], « multicomputer » figure 3.7.

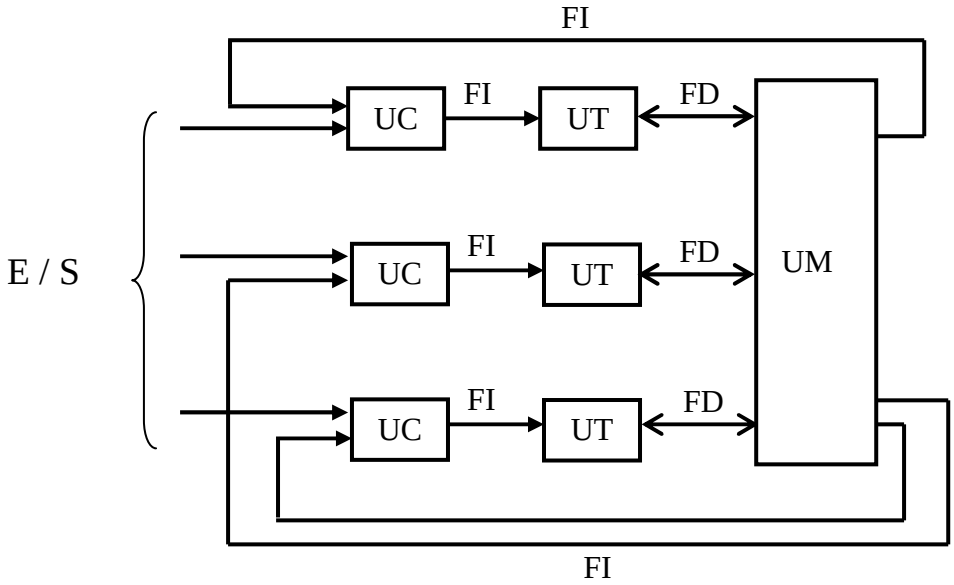


Figure.3.6. Architecture MIMD à mémoire partagée.

Dans ce type d'architectures, un réseau d'interconnexion est nécessaire pour échanger les informations entre processeurs.

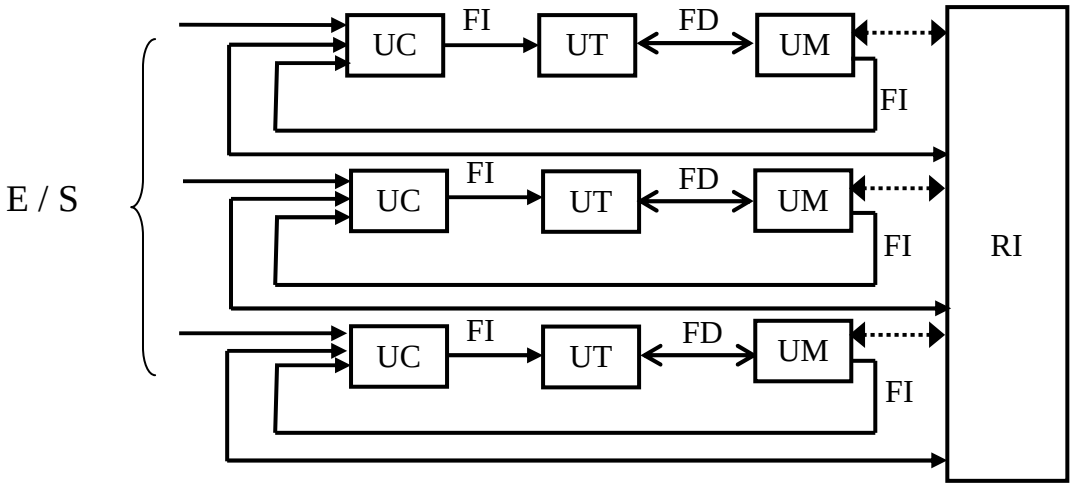


Figure.3.7. Architecture MIMD à mémoire distribuée.

3.5.1.2. Classification de Duncan

3.5.2.1. Architectures synchrones

Leur principale particularité est de posséder une seule horloge et un seul contrôleur de programmes. Le parallélisme peut être obtenu de deux façons, soit en dupliquant les unités de traitement, soit en décomposant les opérations en différentes étapes successives et en les réalisant sur des unités de traitement mise en cascade (pipeline des opérations). Les calculateurs SIMD et architectures systoliques [82] [83] et Qinton [84] font partie de ce type d'architecture. On trouve aussi les systèmes à traitement vectoriel [85] et les systèmes à traitement temporel de recouvrement [86].

3.5.2.2. Architectures asynchrones

Ces architectures sont de type MIMD, pour lesquelles un ensemble de processeurs exécutent des instructions différentes. Une distinction est apportée par le type de communication des données entre processeurs. Les systèmes à mémoire partagée est les systèmes à mémoires distribuée font partie de ce type d'architecture.

3.5.2.3. Architectures mixtes

Deux classes existent pour ce type d'architectures : les architectures flots de données qui ne sont pas contrôlées par une unité spécialisée, mais sont directement contrôlées par les données elles-mêmes [87] et les architectures de type Front d'onde (*Wavefront*). Selon [88], sont un mélange entre architectures systoliques et les architectures flots de données. Les machines à réduction [74] font partie de ce type d'architectures.

3.6. Modèles de programmation parallèle

La parallélisation d'un algorithme peut être définie par la maxime « diviser pour régner ». En effet, si l'on considère un algorithme séquentiel comme une tâche T à effectuer sur des données D, le propre du parallélisme est de définir un ensemble de tâches Ti à partir de T et un ensemble de données Dj à partir de D et de répartir ces tâches et ces données sur les différents nœuds des machines parallèles. Cette division sur les tâches et sur les données a été formalisée par deux modèles de programmation [89]:

- Le modèle de division sur les données est appelé le parallélisme de données. Son principe est d'effectuer des actions successives sur des données réparties sur les processeurs.
- Le modèle de division par les tâches est appelé parallélisme de contrôle. Son principe est de décrire une application comme un ensemble de tâches indépendantes pour pouvoir les effectuer en parallèle.

Pour une parallélisation efficace, une application parallèle issue de l'un de ces deux modèles de programmation doit prendre en compte les problèmes liés à l'équilibrage de charge, au placement des données et au regroupement des résultats.

3.6.1. Parallélisme de données

Le parallélisme de données consiste à découper les données et à les répartir sur les mémoires des processeurs. Deux types de partitionnement sur les données sont envisageables: le partitionnement sur les données initiales ou le partitionnement sur les données résultats. Dans le premier cas, on distribue l'espace des données initiales en sous-domaines sur chaque processeur; dans le second cas, chaque processeur est responsable d'un sous domaine de l'espace résultat.

Dans la plupart des applications, les espaces de données initiales et de données résultats peuvent être divisés par un découpage similaire, ce qui simplifié la parallélisation.

Après la répartition, chaque processeur effectue un calcul séquentiel sur les données dont il est propriétaire. Le séquencement des instructions peut être soit centralisé, soit distribué:

- Dans le cas d'un séquencement centralisé, le parallélisme de données a pour modèle d'exécution les machines SIMD. En effet, ce type de machine est par essence une machine « *data parallel* », car une même instruction est effectuée sur des données réparties sur les mémoires locales.
- Dans le cas d'un séquencement distribué, on utilise le modèle de programmation SPMD « Single Programme Multiple Data ». Ce modèle distribue à chaque nœud une copie d'un programme. Ce programme est constitué de tâches de calcul et de tâches de communication. Contrairement aux machines SIMD où chaque instruction est

synchronisée, ce modèle effectue les tâches de calcul en parallèle de manière asynchrone, ce qui rend la parallélisation potentiellement plus efficace. Les tâches de communication servent à la synchronisation des calculs et aux transferts éventuels des données. L'équilibrage des tâches de calcul reste l'un des principaux problèmes des applications développées sur le modèle SPMD.

Si on découpe le programme en un ensemble fini de tâches T_i, alors chaque processeur effectue la même tâche T_i sur des données différentes D_1, D_2, …, D_j (figure 3.8).

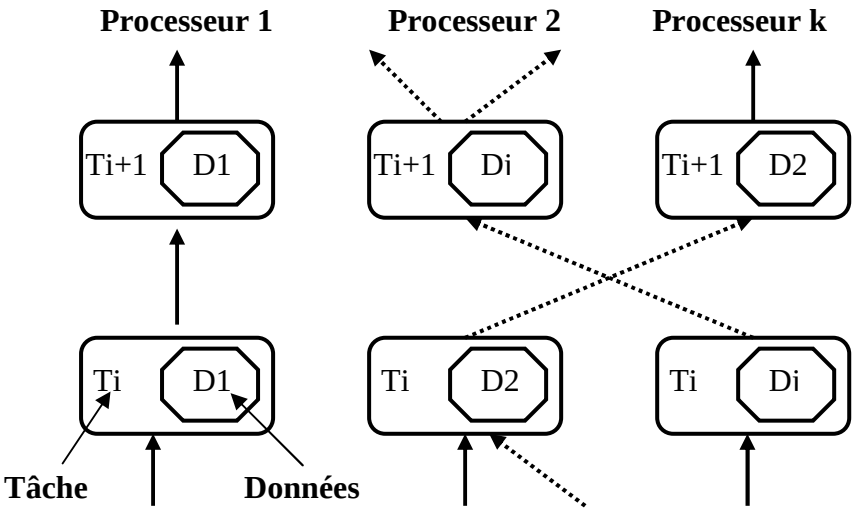


Figure.3.8. Parallélisme de données

3.6.2. Parallélisme de contrôle

Le parallélisme de contrôle définit un programme parallèle comme étant l'expression d'un algorithme sous la forme d'un ensemble de calculs pouvant être effectués en parallèle. On formalise cet ensemble de calcul par un ensemble de tâches. Chaque tâche est un programme séquentiel classique auquel on ajoute des primitives de communication qui définissent les interactions d'une tâche avec les autres. L'algorithme peut être représenté sous la forme d'un graphe orienté où les nœuds formalisent les tâches et les arcs orientés, les communications. Dans ce cas, les processeurs effectuent des tâches différentes sur des données différentes.

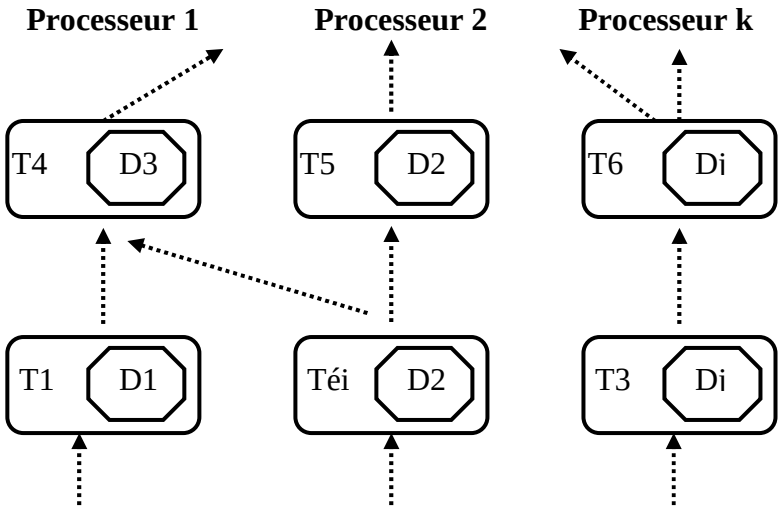


Figure.3.9. Parallélisme de contrôle

Le principal problème de cette classe d'applications est le placement optimal des tâches sur les processeurs d'une machine parallèle. Dans le cas idéal, on souhaiterait disposer d'une machine parallèle composée d'autant de processeurs qu'il y a de tâches. Ce type de parallélisme est implémenté sur les machines de type MIMD.

3.7. Etapes de développement d'une application exécutée sur une architecture parallèle

Le développement d'une application sur une architecture parallèle peut être décomposé en trois étapes, correspondant à des degrés croissant de prise en compte des caractéristiques de la machine cible.

Première étape : Analyse et programmation

La première étape consiste en une analyse de l'application sur sa traduction sous la forme d'un programme parallèle sans tenir compte de la taille de la machine cible.

L'application sera au contraire analysée le plus finement possible de manière à détecter toutes les opportunités de parallélisme qu'elle cache.

Reléguer la prise en compte des caractéristiques physiques de la machine cible aux étapes ultérieures présentes un triple intérêt :

- Évaluation de l'adéquation entre l'application et une mise en œuvre sur un architecture parallèle permet d'éviter un travail inutile de codage au cas où le gain (potentiel) de performances s'avèrerait insuffisant et de déterminer la taille maximale de la machine cible (dans le cas favorable),
- Préservation du travail de codage de l'application en cas d'évolution du cahier des charges imposant un redimensionnement de la machine et

- Une plus grande liberté dans la manière de répartir le travail entre les différents processeurs.

La structure du programme peut être condensée sous la forme d'un graphe (des flux) inter-tâches dont chaque sommet correspond à une tâche et chaque arc à un flux de messages entre les deux tâches qu'il relie. Pour le travail de placement, les sommets et les arcs de ce graphe seront pondérés par des estimations des charges de calcul et des volumes de messages à échanger.

Si nous remplaçons chaque sommet par un processeur et chaque arc par un lien, un tel graphe décrit de fait une machine virtuelle la plus rapide possible pour l'application.

L'analyse et la programmation effectuées, l'utilisateur prend en compte les critères de coût et de performances du cahier des charges et détermine la taille de la machine physique à utiliser.

Les deux étapes suivantes consistent alors à projeter la machine virtuelle définie par le programme sur cette machine cible.

Deuxième étape : Placement des tâches

Dans un premier temps, chaque tâche du programme parallèle doit être affectée à un nœud de la machine cible : cette opération est connue sous le nom de placement des tâches sur les processeurs.

La recherche d'un placement optimal (temps d'exécution minimal) est une opération délicate qui doit à la fois tenir compte de la durée des calculs réalisés par les tâches et des délais de communication.

Le placement se décompose en fait en deux opérations :

- Le regroupement des tâches à placer ensemble sur un même nœud, c'est-à-dire la contraction du graphe des tâches en un graphe (des flux) inter-processeurs, de degré égal à la taille de la machine cible.
- Le placement proprement dit, c'est-à-dire l'association biunivoque des groupes de tâches ainsi formés aux processeurs de la machine cible.

Troisième étape : Projection des flux de communication

Notons que si toutes les tâches communiquent et se synchronisent deux à deux, le réseau d'interconnexion de cette machine devient un Réseau Totalement Maillé (RTM), c'est-à-dire tel qu'il existe un lien direct de tout processeur vers chacun des autres ; tous les processeurs étant donc directement voisins.

Si l’étude de liaisons entre tâches ne fournit qu'un nombre limité de liaisons bipoints entre les processeurs, au dessus desquelles le graphe des flux inter processeurs devra être reconstruit par multiplexage.

Le multiplexage entraîne une vitesse de communication non uniforme, qui va dépendre de la position relative des deux nœuds.

Le problème du placement optimal des tâches s'avère ainsi très complexe, puisque ce placement à la fois dépend et se répercute sur les coûts de communication entre les nœuds.

3.8. Conclusion

Nous avons présenté d'abord dans ce chapitre une vue d'ensemble du parallélisme et une présentation des différentes classes des architectures parallèles, ceci nous a permis de présenter les principaux modèles d'exécution et de programmation. Ces modèles forment la base de toute application parallèle. Un programmeur doit construire son application en fonction des caractéristiques d'un des modèles de programmation parallèle.

Nous avons évoqué dans les chapitres précédents, le problème de la recherche d'information de routage dans un routeur IP. On peut noter que la plupart des solutions sont des algorithmes séquentielles utilisent des structures de données arborescentes. La dimensionnalité des tables de routage (des millions de préfixes) est un frein pour la manipulation des tables de routages de tailles réelles. L'approches parallèle, à été retenu pour réduire ce problème par l'utilisation d'un parallélisme de données. Dans le chapitre suivant, nous présenterons notre approche parallèle basée sur le modèle de parallélisme de données pour la recherche d'informations de routage dans un routeur Internet.

Chapitre 4

L’architecture parallèle « PARIR »

Dans ce chapitre, nous allons tout d’abord présenter un algorithme de décomposition de la table de routage en plusieurs sous tables équilibrées en nombre d’entrées, afin de les manipuler en parallèle, ensuite nous présentons notre algorithme parallèle de recherche des plus long préfixes dans les différentes sous tables de routage. Nous y présentons aussi l’architecture parallèle proposée pour l’exécution de l’algorithme parallèle, et analysons la performance de cette solution dans différents scénarios. Nous montrons que l’efficacité de cette architecture dépend du nombre de sous tables de routage. Enfin nous décrivons et simulons cette architecture par le langage de description du matériel (VHDL).

4.1. Introduction

Toutes les fonctions d’un routeur exigent des temps de réponse rapides. L’opération de recherche d’information de routage dépend aussi du nombre d’entrées dans les tables de routage qui croit exponentiellement. Pour accélérer cette opération, nous proposons de disposer d’un système parallèle dédié à la recherche d’information de routage dans les tables de routage des routeurs IP.

La première partie de ce chapitre présente une description de la conception des systèmes électroniques assistée par l’ordinateur. La suite du chapitre est consacrée à la description, la modélisation et la validation par simulation de notre architecture proposée.

Nous proposons un algorithme de décomposition de la table de routage principale en plusieurs sous tables (parallélisation des données), chaque sous table contient des préfixes expansés à partir des préfixes originaux (de la table de routage originale) ayant des longueurs qui appartiennent à un intervalle [E1, E2]. Nous avons proposé aussi un algorithme parallèle de manipulation des sous tables de routage générées précédemment. Enfin, nous discutons l’implémentation de notre algorithme par une architecture parallèle « PARIR » (***P****arallel*

***A**rchitecture for **R**outing **I**nformation **R**esearch*), qui manipule des sous tables de routage en parallèle. Notre schéma emploie un système multi modules pour l'opération de recherche parallèle dans les différentes sous tables de routage, ce qui est la clef de notre proposition pour accélérer l'opération de recherche du plus long préfixe correspondant dans la table de routage décomposée.

Nous avons ensuite développé dans ce chapitre la description du prototypage de notre architecture. Considérant la complexité du système, nous avons choisi une description basée sur un langage matériel de haut niveau VHDL, qui présente comme avantage une grande flexibilité de conception. En fin de ce chapitre, nous testons nos modèles par simulation.

4.2. Conception assistée par ordinateur

Un circuit intégré (*integrated circuit*) réalise une fonction électronique sous la forme d'un ensemble de composants électroniques miniaturisés assemblés sur un même substrat, usuellement de silicium. Les progrès constants des techniques de fabrication permettent aujourd'hui de placer plusieurs dizaines de millions de transistors sur une surface de silicium très petite.

En analysant l'évolution de la production industrielle d'ASICS (*Application Specific Integrated Circuit*) [90] ou de FPGA (*Field Programmable Gate Array*), on constate que ceux-ci, bénéficiant des progrès technologiques, et sont de plus en plus complexes. On sait intégrer à l'heure actuelle sur silicium des millions de portes pouvant fonctionner à des fréquences supérieures à 600 MHz. En effet, plus de 80% des ASICs futurs comporteront un ou plusieurs microprocesseurs.

Par ailleurs, si on considère qu'un ingénieur confirmé valide 100 portes par jour, il lui faudrait 500 ans pour un projet de 12 millions de portes et ceci pour un coût de 75 millions de dollars [91]. Ceci parait totalement absurde, mais pourtant réalisable, cela est possible grâce à l'évolution des méthodes de CAO.

Il existe trois grands types de méthodes de conception: Les méthodes descendantes, les méthodes ascendantes et les méthodes mixtes.

Les méthodes descendantes (*top-down*) sont basées sur une suite de raffinements successifs partant d'un cahier des charges pour aboutir à une description détaillée de la réalisation. Le cahier des charges définit principalement les fonctions à réaliser et les conditions dans lesquelles ces fonctions devront s'exécuter. Les méthodes descendantes sont bien adaptées à la réalisation de circuits dont la structure peut être optimisée de manière très flexible à partir d'un ensemble de cellules standard (*standard cells*). Les contrôleurs (séquenceurs) sont des exemples typiques de tels circuits.

Les méthodes ascendantes (*bottom-up*) se basent sur l'existence de modules (primitives ou fonctions plus complexes) caractérisés, c'est-à-dire dont les fonctions et les performances sont connues. Une réalisation possible est alors construite par assemblage à l'aide d'un processus de sélection de modules. Le processus est tel qu'il doit garantir que les choix faits satisfont les contraintes imposées par le cahier des charges. Les méthodes

montantes sont bien adaptées à la réalisation de circuits dont la structure est essentielle à leur bon fonctionnement. Les circuits réguliers (mémoires, chemins de données (*datapath modules*)) en sont des exemples types. Un additionneur, bloc de base très courant dans les circuits intégrés, peut par exemple être réalisé selon différentes architectures avec différents niveaux de performances.

Les méthodes mixtes (*meet-in-the-middle*) sont une combinaison de méthodes descendantes et de méthodes ascendantes. Elles sont particulièrement adaptées à la réalisation de circuits à applications spécifiques (ASIC) possédant un grand nombre de composants personnalisés comme des multiplieurs, des unités de contrôle et de la mémoire.

Le processus de conception passe ainsi par un certain nombre d'étapes, chacune d'elles nécessitant une description de l'état du système sous forme graphique (diagrammes, schémas, etc.) ou textuelle (algorithmes). Ces descriptions peuvent être fournies par le concepteur ou produites par des outils logiciels (langages de description de matériel).

4.3. Modèles

La création d'un modèle résulte d'un processus de structuration d'un ensemble de connaissances, parfois expérimentales, que l'on dispose à propos d'un phénomène ou d'un système physique. Un modèle peut représenter le comportement et/ou la structure d'un système donné.

Le comportement (*behavior*) d'un système se concentrer sur la (les) fonction(s) du système en exprimant des relations de cause à effet. Les fonctions d'un système peuvent être organisées de manière hiérarchique (fonctions imbriquées ou récursives).

La structure (structure) se concentre sur la manière dont un système est organisé hiérarchiquement en sous-systèmes. La structure d'un système peut être de plus considérée sous deux aspects complémentaires: un aspect adimensionnel pour lequel les notions de dimension, de taille et de forme sont ignorées et un aspect géométrique qui tient compte de ces notions. Une description structurelle adimensionnelle est usuellement un schéma, éventuellement hiérarchique, représentant uniquement les liens topologiques entre éléments du système.

4.4. Synthèse et Simulation

Un modèle est une représentation abstraite d'une réalité physique dont on ne conserve que les aspects essentiels à une certaine utilisation. Un modèle ignore donc les détails, soit parce qu'ils sont inutiles ou peu importants, soit parce qu'ils ne sont pas encore connus. Dans le premier cas, le modèle sert de support à l'analyse par simulation, soit l'extraction et la vérification de propriétés. Dans le second cas, le modèle sert de support à la synthèse, soit la dérivation d'un modèle plus détaillé en fonction de contraintes de conception.

4.4.1. Validation et vérification de description HDL

La vérification qu'aucune erreur n'a été introduite entre le niveau algorithmique et le niveau RTL reste donc une issue importante qui exige une automatisation. Ce domaine est connu sous le nom de validation fonctionnelle ou encore vérification fonctionnelle. L'IEEE [92] définit l'activité de vérification comme le moyen d'établir la correspondance entre un produit et sa spécification, et la validation comme le moyen d'assurer que le produit accomplit bien la fonction pour laquelle il a été conçu. En général, le terme de vérification fait référence aux méthodes consistant à « démontrer » qu'un circuit va se comporter comme il est souhaité, alors que le terme de validation fait référence aux méthodes consistant à exciter une description de circuit par une série de stimuli. On peut donc distinguer deux approches :

- les approches de type vérification formelle [97];
- les approches basées sur la simulation de la description sous test.

Malgré les progrès accomplis, la vérification formelle qui propose de prouver mathématiquement l'exactitude d'une description n'est seulement réalisable que pour de petites descriptions. En effet l'automatisation de la méthode implique l'analyse exhaustive d'un espace d'état très large et est donc restreinte à des parties réduites d'une description.

De ce fait, la validation basée sur la simulation reste encore la meilleure méthode pour la vérification de conception.

4.4.1.1. Simulation fonctionnelle

Tester un circuit revient à lui appliquer des stimuli, des vecteurs de tests, et à analyser sa réponse. A un deuxième niveau, le test inclut l'environnement dans lequel le circuit est introduit, la carte sur laquelle il est implanté. Les conditions expérimentales reproduiront, dans un premier temps, les conditions de fonctionnement normales ; dans un deuxième temps elles chercheront à cerner les limites, une fréquence maximum, par exemple.

Tester un module logiciel revient à écrire un programme qui lui transmet des données, reçoit les résultats et les analyse. Dans une première étape les données transmises sont celles attendues, elles servent à identifier d'éventuelles erreurs ; dans une deuxième étape, les données sortent de l'ordinaire prévu, provoque une situation de faute.

Le test est une opération difficile à mener [93]. Or le but du test est de chercher la faute, même, et surtout, la faute maligne, celle qui ne se révèle pas au premier abord (une situation de débordement dans un calcul numérique, par exemple).

VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) est un langage qui décrit et teste des circuits (IEEE 1076-1993). Les programmes de test simulent l'environnement d'un montage, ils bénéficient de la souplesse des langages informatiques pour recréer virtuellement toutes les situations expérimentales possibles et imaginables, même celles qui seraient difficile à réaliser en pratique.

La simulation d'une description VHDL se décompose en deux phases [93]: une phase d'initialisation suivie par l'exécution répétitive des processus de la description de ce modèle. Chacune de ces répétitions est appelée cycle de simulation. À chaque cycle, les valeurs de

tous les signaux dans la description sont calculées. Si suite à ce calcul, un événement se produit sur un signal donné, les processus qui sont sensibles à ce signal, reprennent et sont exécutés en tant qu'éléments du cycle de simulation.

La phase d'initialisation

Le simulateur VHDL effectue un cycle de simulation durant lequel tous les processus de la description sont actifs. Cette phase permette d'établir l'état initial du système. Les valeurs de départ des signaux et des variables sont déterminées en se reportant aux règles suivantes :

- La valeur initiale d'un signal (ou d'une variable) est spécifiée dans la déclaration (exemple : signal a : bit <= '0').
- La valeur du signal n'est pas spécifiée, auquel cas, cette valeur est implicite : la première valeur du domaine de l'ensemble de définition.
- À la fin de ce cycle de simulation toutes les données de la description VHDL ont été initialisées. La phase d'exécution peut alors débuter.

La phase d'exécution

C'est au début de cette phase qu'est introduit le vecteur d'entrée. La simulation de la phase d'exécution se fait par scrutation des signaux sensibles, déterminant quels processus sont actifs et lesquels sont suspendus. La phase d'exécution peut comporter plusieurs cycles de simulation. Elle arrive à son terme lorsque tous les processus sont suspendus.

4.4.2. Synthèse

Le concepteur du circuit dispose de spécifications textuelles du circuit (cahier des charges) qui lui permettent de décrire, à l'aide d'un langage de description de matériel (tel que Verilog ou VHDL), son comportement.

Cette description se présente sous la forme d'une description algorithmique. La transformation de cette première description en une description au niveau RTL est appelée synthèse architecturale (ou comportementale).

Le synthétiseur interprète un programme (description de haut niveau) et en déduit une structure de schéma (description à un niveau plus bas) [94]. Dans une deuxième étape, il cherche à reproduire ce schéma dans une technologie, au moindre coût (surface de silicium), avec l'efficacité la plus grande (vitesse maximum).

Ce processus de synthèse est l'application d'une méthodologie par le biais des techniques de conception. Le langage VHDL permet à la fois de valider une conception, de construire les programmes de test utiles et de générer automatiquement des schémas d'implantation sur le silicium [93].

La réalité du monde physique introduit des différences de comportement entre le modèle fonctionnel et sa réalisation dans un circuit.

Une fois le programme compilé, traduit en un schéma de portes et de bascules interconnectées, il est intéressant de prévoir, compte tenu de la technologie employée, les limites de son fonctionnement.

4.5. Langages de description de matériel

Au début des années 90, l'augmentation du nombre de portes de circuits numériques a ouvert la voie aux langages de description matérielle de haut niveau (les HDL, ***H****ardware* ***D****escription* ***L****anguage*). Deux d'entre eux ont émergé et sont couramment utilisés : VHDL (*VHSIC* ***H****ardware* ***D****escription* ***L****anguage*) et Verilog.

Un langage de description de matériel (HDL: *Hardware Description Language*) est un outil de description, éventuellement formel, permettant la description du comportement et de la structure d'un système matériel. Un langage de description de matériel idéal a les propriétés suivantes [94]:

- Il supporte la description d'une large gamme de systèmes, logiques (numériques) et analogiques.
- Il permet la description de l'état de la conception pour toutes les étapes du processus. Le fait d'utiliser un langage unique renforce la cohérence entre différentes représentations d'un même système.
- Il renforce aussi la cohérence des outils logiciels utilisés pour la simulation et la synthèse. Il peut être directement compris comme langage d'entrée pour de tels outils.
- Il est indépendant de toute méthodologie de conception, de toute technologie de fabrication et de tout outil logiciel. Il permet au concepteur d'organiser le processus de conception en fonction des besoins (conception descendante - *top-down design*, conception ascendante - *bottom-up design*).
- Il supporte plusieurs niveaux d'abstraction et autorise des descriptions hiérarchiques
- Il est extensible: il permet au concepteur de définir de nouveaux types d'objets et les nouvelles opérations correspondantes.
- Il est standardisé par l'intermédiaire d'organisations reconnues comme l'IEEE, l'ANSI ou l'ISO. Ceci favorise une large acceptation à la fois de la part des fournisseurs d'outils et des différentes communautés de concepteurs.

4.6. Parallélisation des données par la décomposition de la table de routage

Par l'analyse de la table de routage originale, on introduit une méthode de décomposition de la table de routage en plusieurs petites sous tables en se basant sur la longueur des préfixes, et ensuite nous étendons les préfixes de chacune des sous tables pour rendre tous les préfixes de même longueurs afin de simplifier et accélérer la procédure de recherche du plu long préfixe correspondant dans les différente sous tables. Les sous tables de routage sont de petites tailles (en nombre d'entrées) par rapport à la table de routage originale ce qui nous permettent de faire une recherche par comparaison directe d'une partie de l'adresse IP avec les préfixes de la sous table.

4.6.1. Terminologie et définitions

Nous présentons dans cette section des définitions utilisées dans le reste du chapitre.

Définition.4.1. Groupe de préfixes : Un groupe de préfixes est identifié par un intervalle des longueurs des préfixes ayant une extrémité initiale E1 et une extrémité terminale E2. Chaque groupe ne contient que des préfixes de la table originale ayant une longueur L tel que : $E1 \leq L \leq E2$. La valeur E2 définie la longueur des préfixes expansés des sous table de routage associée à ce groupe de préfixes.

Définition.4.2. Sous table de routage : Une sous table des préfixes expansés est dérivée de la table de routage originale en augmentant chaque préfixe de longueur L vers 2^{LG-L} préfixes de longueur LG. Exemple : Le préfixe « 10.0.0.0/8 » est expansé dans un groupe de longueur maximale LG =12 vers $2^{12-8} = 16$ préfixes. Alors chacune des sous tables ne contiendra que des préfixes de même longueur égale à la longueur maximale du groupe.

Définition.4.3. Ajustement des intervalles de deux groupes adjacents : Soit deux groupes adjacents G1 et G2 correspondant aux intervalles des longueurs des préfixes [E1, E2] et [E2+1, E3] respectivement. L'ajustement de ces intervalles génère deux nouveaux intervalles: [E1, E2-P] et [E2-P+1, E3] respectivement ou bien [E1, E2+P] et [E2+P+1, E3] respectivement ; tel que P c'est le pas d'ajustement donné.

4.6.2. Algorithme de décomposition de la table de routage

Notre algorithme sert à décomposer la table de routage en N sous tables de tailles équilibrées (tailles proches), en se basant sur les longueurs de préfixes. Chaque sous table ne contient que les préfixes de la table de routage principale ayant des longueurs qui appartiennent à un intervalle de longueurs associé à cette sous table. Les intervalles de longueurs sont générés automatiquement avec l'objectif d'équilibrer les tailles des différentes sous tables.

Tous les préfixes d'une sous table sont expansés ensuite vers des préfixes de même longueur (longueur maximale de la sous table), ce qui simplifie la recherche du port de sortie par comparaison directe entre une partie de l'adresse IP et les préfixes de la sous table.

Paramètres de l'algorithme :

- Longueur de l'adresse IP : 32 ou 128 ;
- Nombre de groupes : N = 2, 3, 4, …
- Différence minimale entre les nombres des préfixes expansés correspondants aux deux groupes adjacents.
- Pas d'ajustement des intervalles des groupes de préfixes.

L'algorithme :

1- Initialement générer N intervalles de mêmes longueurs à partir de l'intervalle original.

2- Distribuer les préfixes originaux sur les N groupes correspondants aux N intervalles.

3- Calculer les nombres de préfixes expansés de chacun des groupes.

4- Evaluer la différence maximale D entre les nombres de préfixes expansés correspondants aux groupes adjacents.

5- Si D < Différence minimale donnée comme paramètre, alors arrêter les calculs et aller à l'étape 6.

Sinon Ajuster les intervalles des deux groupes adjacents correspondant à cette différence D et aller à l'étape 2

6- Générer les N sous tables de routage correspondant aux N groupes générés précédemment.

4.6.3. Procédure d'expansion des préfixes

La procédure d'expansion des préfixes est très simple ; par exemple, soit la table de routage présentée dans le tableau 4.1. L'expansion des préfixes de cette table vers des préfixes de longueur 4 (longueur maximale des préfixes du tableau 4.1) nous engendre les préfixes présentés dans le tableau 4.2.

Préfixe	**Longueur du préfixe**	**Pas suivant (*Next Hop*)**
11*	2	A
10*	2	B
111*	3	C
1111*	4	D

Tableau.4.1 : exemple 4.1 d'une table de routage des préfixes originaux

Le préfixe (11*) est expansé vers 4 entrées : 1100 ; 1101 ; 1110 et 1111 ; L'entrée (1111) existe déjà dans la table de routage, mais les entrées (1100) ; (1101) et (1110) n'existent pas, alors il faut les ajouter dans la table de routage des préfixes expansés.

Préfixe	**Longueur de préfixe**	**Pas suivant (*Next Hop*)**
1100 (11*)	4	A
1101 (11*)	4	A
1110 (11*)	4	A
1000 (10*)	4	B
1001 (10*)	4	B
1010 (10*)	4	B
1011 (10*)	4	B
1110 (111*)	4	C
1111 (1111*)	4	D

Tableau.4.2: Sous table de routage des préfixes expansés de longueur 4

Après l'expansion des préfixes, chacune des sous table de routage ne contiendra que des préfixes de même longueur selon le groupe de préfixes. Alors la recherche se fait par une comparaison directe c.-à-d. par un seul accès à la mémoire.

4.6.4. Analyse des résultats

Les résultats suivants sont obtenus par notre algorithme de décomposition d'une table de routage de 100 préfixes de longueur aléatoire avec les paramètres suivants :

- Adresse IP de longueur : 32
- Nombre de groupe : 10 pour la courbe de la figure.4.2
- Différence minimale entre le nombre des préfixes expansés entre tous deux groupes adjacents est : 10
- Pas d'ajustement : 1

La figure.4.1 représente une comparaison entre le nombre de préfixes expansés par notre algorithme d'expansion et le nombre de préfixes originaux (de la table de routage originale) dans les 10 groupes générés par notre algorithme de partitionnement des préfixes sur les groupes ; chaque groupe est caractérisé par un intervalle de longueurs des préfixe [E1 , E2] (représenté dans le tableau.4.3); ces intervalles sont ajustés (ayant des longueurs d'intervalle différentes), afin de donner des sous tables de routage équilibrées en termes de nombre de préfixes expansés.

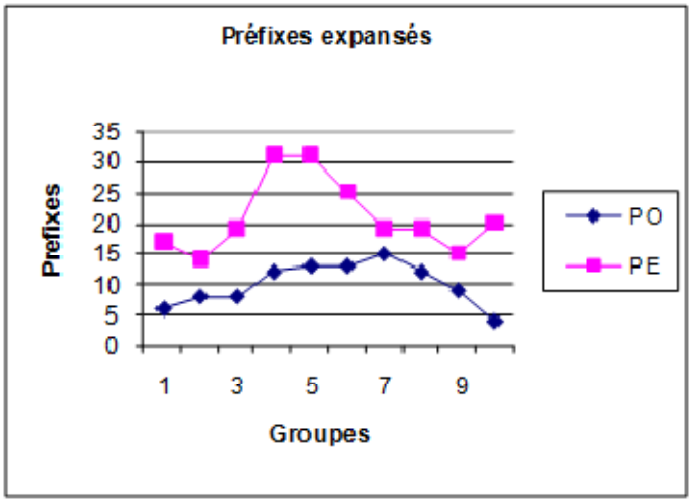


Figure.4.1. Comparaison entre les préfixes expansés et originaux des 10 groupes de préfixes

Dans la figure.4.2 on a représenté le nombre total de préfixes expansés en fonction du nombre de groupes (sous tables de routage), nous remarquons que le nombre total des préfixes expansés est proche du nombre de préfixes originaux, dû au nombre de groupe qui est supérieur à 10 ; alors le cas parfait est le partitionnement de la table de routage originale en un nombre de groupes supérieur à 10.

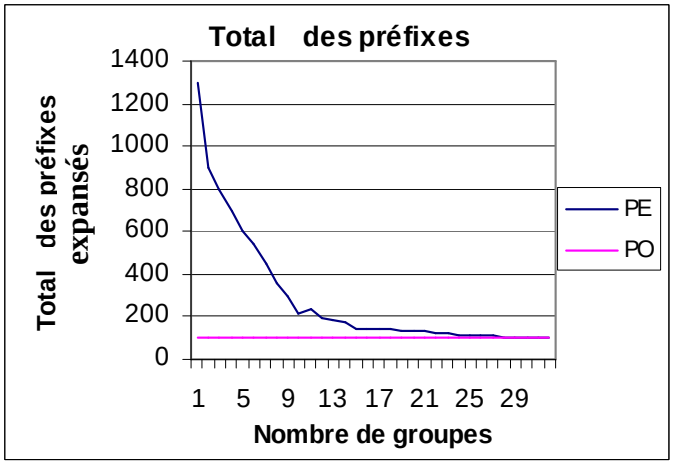


Figure.4.2 : Comparaison entre les préfixes expansés et originaux en fonction du nombre de groupe

Pour le partitionnement de la table de routage en un nombre de sous tables inférieur à 10, le nombre de préfixes générés par l'opération d'expansion est très grand par rapport au nombre de préfixes originaux, ce qui rend le partitionnement non rentable ni en termes d'espace mémoire occupés par les préfixes expansés, ni en termes du temps de recherche du plus long préfixe au niveau de chacune des sous tables à cause de la taille importante de ces sous tables de routage générées par l'algorithme de partitionnement.

Groupe de préfixes	Intervalles initiales			Intervalles après l'ajustement		
	E1	**E2**	**Longueur de l'intervalle**	**E1**	**E2**	**Longueur de l'intervalle**
1	1	4	3	1	5	4
2	5	8	3	6	8	2
3	9	11	2	9	11	2
4	12	14	2	12	14	2
5	15	17	2	15	17	2
6	18	20	2	18	20	2
7	21	23	2	21	22	1
8	24	26	2	23	25	2
9	27	29	2	26	28	2
10	30	32	2	29	32	3

Tableau.4.3. Intervalles des groupes

4.7. Structure de données et algorithme parallèle

4.7.1. Structure de données

La structure de données utilisée pour implémenter la table de routage est le facteur le plus important dans les algorithmes de manipulation des tables de routage. Plusieurs algorithmes de recherche rapide du plus long préfixe sont proposés [19][25][12][29][30][31], mais la plupart sont basés sur des structures arborescentes pour représenter les préfixes de la table de routage; Les arbres sont des structures très difficiles à implémenter en matériel à cause de l'hiérarchie de la structure d'arbre qui demande toujours des traitements récursifs. De tels algorithmes réduisent l'utilisation de la mémoire au détriment du nombre de consultations de la mémoire.

Notre approche utilise les tables d'hachage, puisque généralement, le hachage est simple et facile à implémenter en matériel, mais il a besoin de mémoires de grande taille ; Le problème des tailles de ces mémoires est pris en considération par la décomposition de la table de routage principale en plusieurs sous tables de routage de petites tailles par rapport à la table originale.

4.7.2. Algorithme parallèle

Notre algorithme se déroule en deux étapes principales, durant la première étape, l'algorithme extrait un préfixe à partir de l'adresse IP (une partie de l'adresse IP). Pour chacune des sous tables, la longueur de chaque préfixe extrait à partir de l'adresse IP est égale à la longueur des préfixes de leur sous table. L'algorithme recherche ensuite en parallèle, dans les différentes sous tables de routage, les préfixes correspondants au préfixe de l'adresse IP. Cette recherche est assurée par la fonction *Recherche_PS( )*. Les résultats de

cette fonction sont retournés vers un tableau pour êtres utilisés comme des entrées dans la deuxième étape.

La deuxième étape de notre algorithme est l'étape de sélection du plus long préfixe parmi les données du tableau résultat de la première étape. Cette sélection est assurée par la fonction *Selection_PS( ).*

Entrée de l'algorithme : L'adresse IP de destination du paquet à router (*dest_IP*)

Sortie de l'algorithme : Port de sortie du routeur (*PS*)

L'algorithme :

```
/* Fonction Recherche_Port_Sortie */
String Recherche_PS(dest_IP,STRi, Long_Pref)
{
/* Les paramètres sont : l'adresse IP de destination, la sous table de routage STRi et la longueur des préfixes de cette STRi */
/* Extraction du préfixe à partir de l'@IP  de destination */
Pref_Dest_IP = ""
Pour ( j = 1; j ≤ Long_Pref;  j++)
{
    Pref_Dest_IP  =  Pref_Dest_IP   + dest_IP[j]
}
/* Recherche du port de sortie PS de Pref_Dest_IP dans la sous table STRi */
PS = 0 ;  j = 1 ;
Tant Que  non fin de la table STRi
    Si (STRi [j, Prefixe] = Pref_Dest_IP)
      {    PS = STRi[j, Next_Hop];        break;      }
   Aller à l'entée suivante de la table STRi;
   J++;
}
Retourner (PS);
}

/* Fonction Selection_Port_Sortie */
String Selection_PS(Tab_PS)
{
/* Le paramètre est un tableau Tab_PS qui contient les PS générés en parallèle */
L = 0 ;   P_S = ""
Pour (j = 1 ; j ≤ K ;  j++)
{
```

```
   Si  Tab_PS[j].Long > L
     {
      P_S = Tab_PS[j]
      L= Tab_PS[j].Long
      }
}
Retourner (P_S);
}
```

/* Algorithme principal */

Initialiser K / le nombre d'unités de recherche parallèle */*

```
   Pour (i = 1 jusqu'à K ) Faire en parallèle
   {
    Tab_PS[i].PS = Recherche_PS(dest_IP,STRi,  Long_Pref) ;

    Tab_PS[i].Long = Long_Pref ;
   }
Port_Sortie = Selection_PS(Tab_PS)
}
```

/ Fin de l'algorithme */*

Les structures de données de l'algorithme précédent sont les suivantes

1) **K** : Le nombre d'unités de traitement parallèle (K est un paramètre donné).

2) **STRi** : Une sous table de routage *i* constitué des colonnes suivantes :

- Préfixe
- Next_Hop (le pas suivant ou port de sortie)

3) **Tab_PS** : Un tableau de *K* éléments où chaque élément est structuré comme un enregistrement de deux champs :

- PS : le port de sortie.
- Long : la longueur du préfixe correspondant au PS.

4.8. L'Architecture parallèle « PARIR »

4.8.1. Description de l'architecture « PARIR »

L'architecture complète d'exécution de notre algorithme parallèle de recherche du plus long préfixe, indiquant le port de sortie pour une adresse IP d'un paquet entré, est représentée dans la figure.4.3. Nous avons proposé des unités de recherche qui fonctionnent en parallèle pour implémenter la fonction *Recherche_PS( )* de notre algorithme parallèle, et un composant sélecteur pour exécuter la fonction *Selection_PS( ).* Alors chacune des unités de recherche exécute le même code de la fonction *Recherche_PS( )* et passe le résultat au composant sélecteur.

Une fois que l'adresse IP reçue au niveau des différentes unités de recherche (UR), chaque unité de recherche extrait une partie de l'adresse IP (préfixe) de longueur égale à la longueur des préfixes de sa sous table de routage, pour chercher un préfixe correspondant à la partie de l'adresse IP extraite. L'unité renvoie ensuite les informations de routage, à savoir, le port de sortie de ce préfixe (PS_k), et la longueur du préfixe (L_k). Si le préfixe cherché n'existe pas dans la sous table de routage, L'UR renvoie 0.

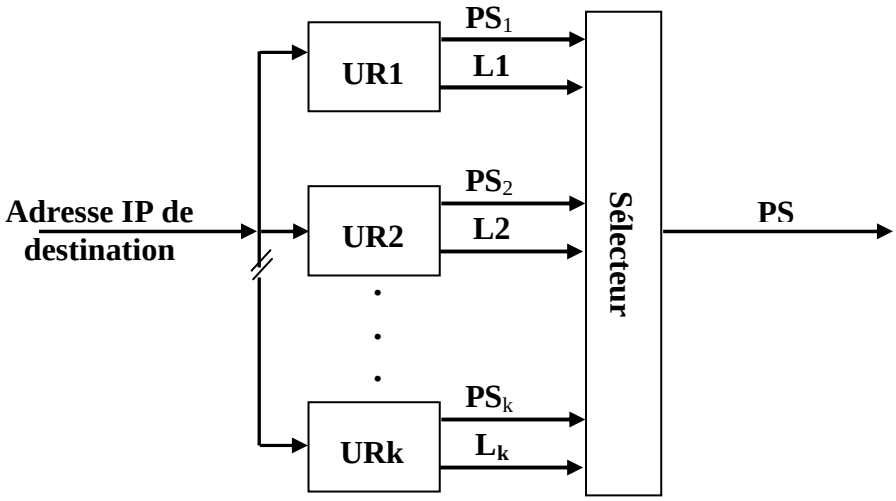


Figure.4.3. Architecture de base

Cette architecture est constituée de deux composants essentiels :

a) Les unités de recherche :

C'est l'unité qui cherche le port de sortie (résultat partiel) dans les sous tables de routage qui contiennent les préfixes associés à cette unité de recherche. Une unité de recherche contient dans sa mémoire locale ses sous tables de routage qui stockent les préfixes de la sous table associée à cette unité de recherche avec ses informations de routage. Une petite mémoire est conçu pour stocker la longueur des préfixes de la sous table de routage associé à cette unité de recherche (tous les préfixes sont de même longueur) et une composante de recherche qui permet d'extraire une partie de l'adresse IP (préfixe) et chercher le préfixe correspondant à cette partie de l'adresse IP dans la sous table de routage et renvoie les informations suivantes : Port de sortie (PS) et une des valeurs 0 ou 1 : 0 si le préfixe

n'existe pas dans la sous table de routage, 1 sinon. L'architecture d'une unité de recherche est représentée dans la figure.4.4.

Toutes les unités de recherche sont similaires puisqu'elles exécutent le même code.

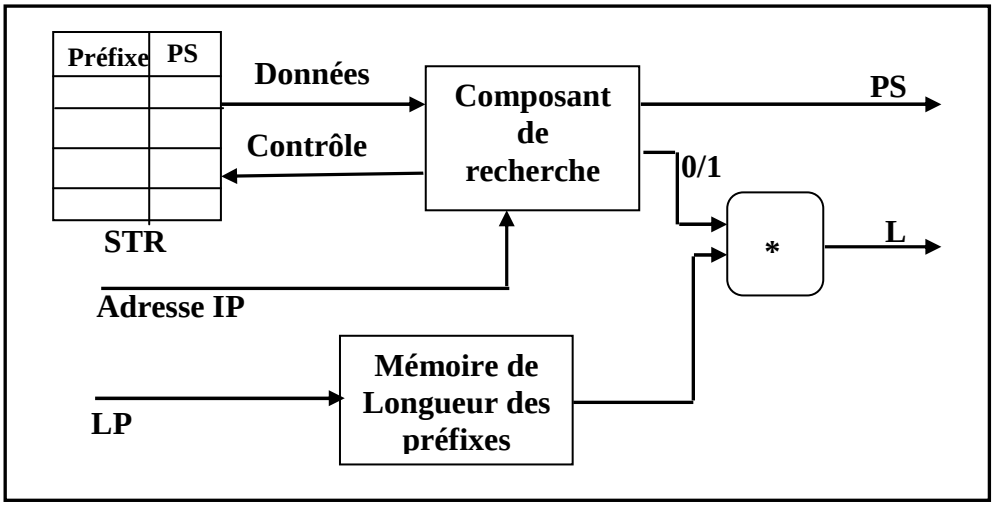


Figure.4. 4. Unité de recherche (UR)

b) Le sélecteur :

Un sélecteur reçoit en entrée l'ensemble des couples (PS_j, L_j) (les sorties des différentes unités de recherche). Il sélectionne parmi ces couples le port de sortie (PS) correspondant à la plus grande longueur L_j. La figure.4.5 représente la structure d'un sélecteur. Le sélecteur est composé de K comparateurs (K est le nombre de sous tables de routage) qui font des comparaisons en parallèle. Chaque comparateur compare une longueur donnée L_j avec toutes les autres longueurs ; si la longueur L_j est supérieure à toutes les autres longueurs, le comparateur renvoie la valeur du port de sortie PS_j entrée, sinon il renvoie 0.

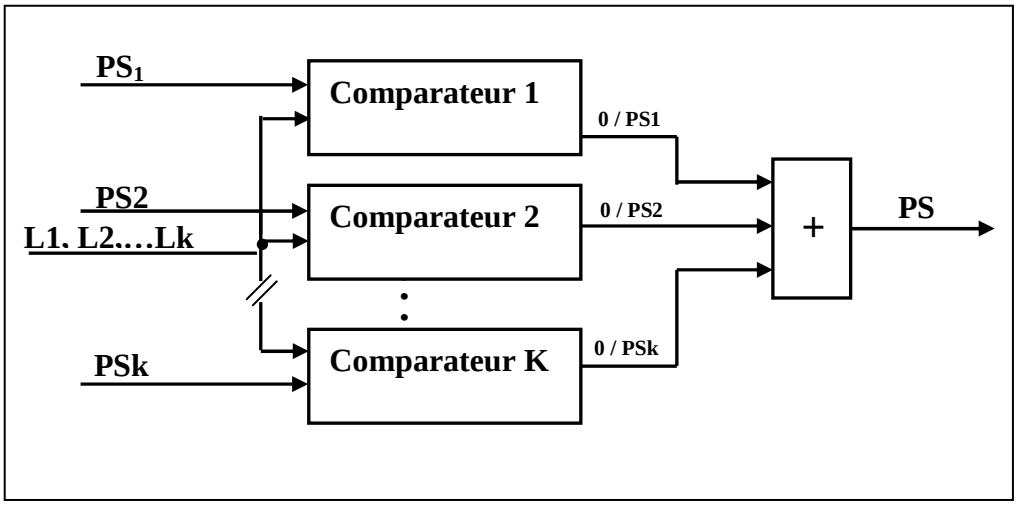


Figure.4.5. Sélecteur

La structure d'un comparateur est représentée dans la figure.4.6. Un comparateur est composé de K opérateurs de comparaison « supérieur », un composant « et logique » et un opérateur d'addition «additionneur ». Tous les comparateurs retournent des valeurs nulles sauf un seul comparateur qui retourne le port de sortie PS correspond à la plus grande longueur. L'additionneur reçoit les valeurs des PS et fait la somme pour donner un seul PS.

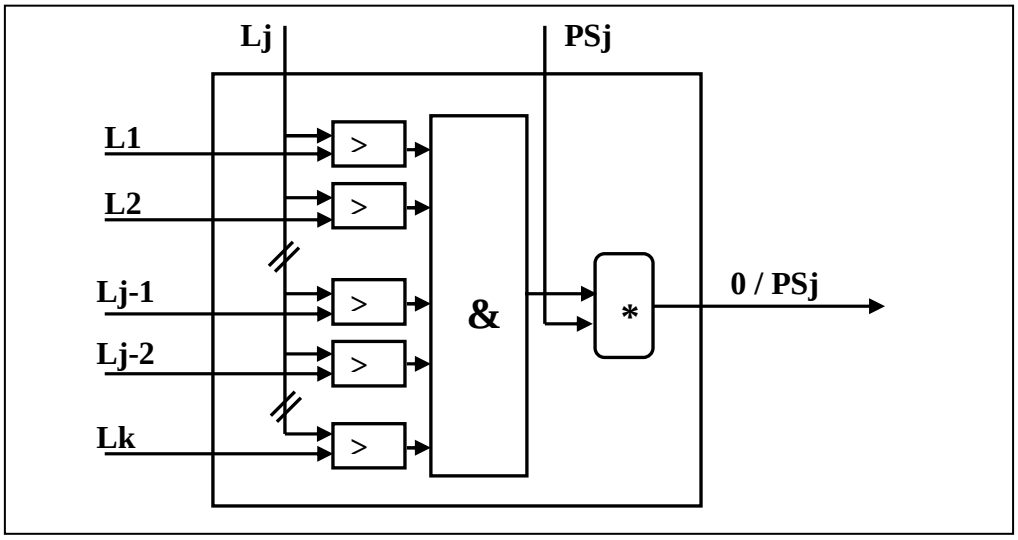


Figure.4.6. Comparateur J

4.8.2. Expérimentations et résultats

Pour analyser la performance de notre architecture proposée, nous définissons deux paramètres de base décrits comme suit :

Le premier paramètre est le temps écoulé pour la recherche du plus long préfixe, dans notre architecture, l'opération de comparaison est l'opération de base. Nous considérons ce paramètre en termes de nombre de comparaisons de l'adresse IP entrée avec les préfixes des sous tables de routage des différentes unités de recherche (URs); étant donnée que l'architecture proposée est une architecture parallèle c.-à-d. que toutes les unités de recherche font des comparaisons en parallèle. Ceci nous permet de prendre le nombre maximum de comparaisons faites par les différentes unités de recherche.

Le deuxième paramètre est l'espace mémoire occupé par les préfixes des différentes sous tables de routage, ce paramètre est évalué en termes de nombre total des préfixes des différentes sous tables de routage.

Pour nos expérimentations, on utilise une table de routage de 5000 préfixes générés aléatoirement. A chaque expérimentation, cette table de routage est décomposée en plusieurs sous tables de routage par notre algorithme de décomposition présenté, le nombre de sous tables doit être égal au nombre d'unités de recherche dans l'architecture.

Nombre d'unités de recherche	Nombre maximum de Comparaisons	Nombre total des préfixes expansés
5	13100	46200
10	2600	13800
15	1600	8500
20	1600	8200
25	1000	6900
30	700	5300

Tableau.4.4. Résultats obtenus

Le diagramme de la figure.4.7 représente d'une part le nombre maximum de comparaisons en fonction du nombre d'unité de recherche (UR) de l'architecture et d'autre part, l'espace mémoire occupé par les préfixes expansés. L'espace mémoire est décrit par le nombre de préfixes expansés des différentes sous table de routage de l'architecture.

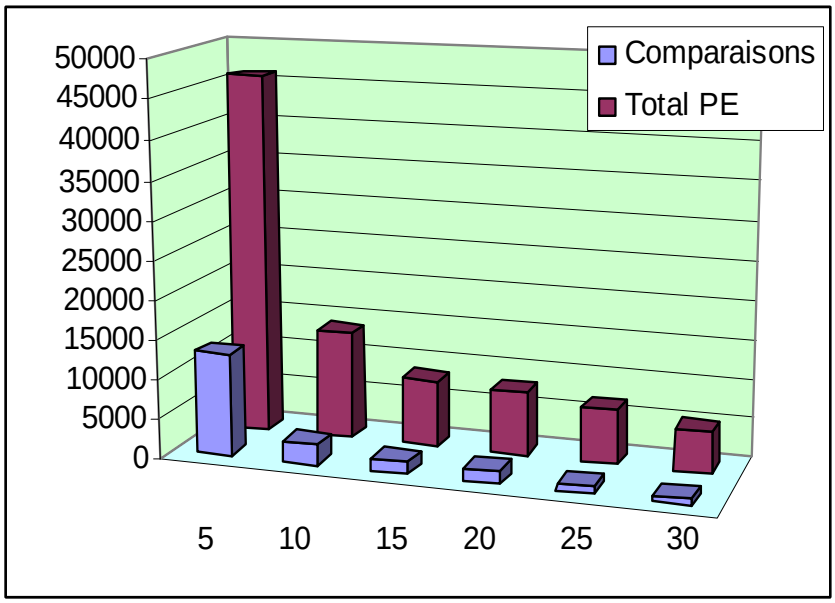


Figure.4.7. Digramme des résultats obtenus

Nous observons dans le diagramme de la figure.4.7 que l'augmentation du nombre d'unités de recherche en parallèle améliore le critère de temps de recherche et aussi le critère d'espace mémoire occupé par les préfixes, puisque l'augmentation du nombre d'unités de recherche augmente le nombre de sous tables de routage. Ceci réduit automatiquement le nombre de préfixes dans les différentes sous tables de routage, ce qui améliore la performance de temps de recherche au niveau de toutes les unités de recherche. Cependant, quand le nombre d'unités de recherche est supérieur à 10, les performances des architectures sont proches comme montre le tableau.4.4.

4.9. Modélisation de notre système

Notre but est d'élaborer une architecture matérielle dédiée à la recherche d'information de routage dans un routeur IP. La modélisation est décrite en langage VHDL indépendamment de la technologie matérielle cible. Nous allons adopter une méthodologie de conception modulaire qui est à base de composants génériques (une bibliothèque de modules réutilisables).

La démarche que nous avons suivie pour décrire et vérifier le circuit est résumée comme suit :

- Analyse et décomposition de notre système en simple modules pour simplifier la description.
- Description en VHDL et validation de chaque module comme suit :
    - Décrire la vue externe par la spécification d'entités de tous les modules.
    - Décrire la vue interne (architecture) de chacun des modules.
- Relier les différents modules entre eux pour réaliser le système global.

4.9.1. Décomposition du système en modules

Nous avons proposé une structure hiérarchique du système à modéliser de telle façon à pouvoir exploiter l'expression du parallélisme offert par le langage VHDL. Les modules de notre système sont organisés comme le montre la figure 4.8.

L'architecture de notre système est constituée de deux composants (modules), à savoir, un « composants de recherche » et un « sélecteur ». Ces deux composants sont reliés et fonctionnent en séquence, puisque le module « sélecteur » reçoit en entrée les résultats du module « composant de recherche ».

Le module « composants de recherche » à son tour est décomposé en N modules appelés « Unité de recherche », toutes les unités de recherches sont similaires. Nous décrivons un seul modèle VHDL pour toutes les unités de recherche, et par l'instanciation, on peut utiliser plusieurs instances de ce modèle. Toutes les unités de recherche à l'intérieur d'un même composant de recherche fonctionnent en parallèle.

Le module « sélecteur » est décomposé aussi en plusieurs modules appelés « comparateur », les modules « comparateurs » sont similaires, c.-à-d. exécutant le même code sur des données différentes, en parallèle.

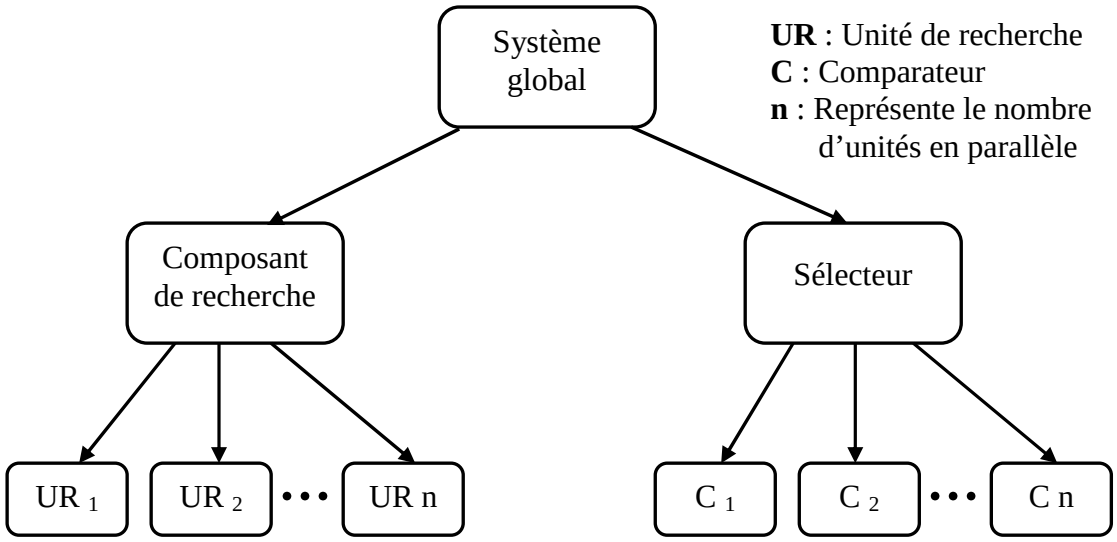


Figure. 4.8. Structure en module de notre système

Nous avons modélisé chaque composant séparément, et nous avons stocké les modules dans une bibliothèque de composants réutilisables afin d'économiser le temps de développement. Les modules de notre système sont développés et testés par simulation, sont exposés dans ce qui suit.

4.9.2. Le langage VHDL

D'un point de vue industriel, les langages de description de matériel électronique sont une nécessité. Ils sont très utilisés. Ils permettent de simuler les circuits avant leur réalisation, de constituer des bibliothèques de modules, de décrire la spécification d'un circuit [95]. La description préalable du comportement d'un futur circuit devient une étape obligée dans son processus de conception. Cette description constitue souvent un élément contractuel. Elle est souvent exigée dans les marchés militaires, aéro-spaciaux, télécommunication, sécuritaire,…

Le langage VHDL (*Very High SpeedIntegrated Circuits Hardware Description Language*) [95], [96] résulte d'un effort conjoint des compagnies intermetrics, IBM et Texas dans les années 80. Quelques années plus tard, en 1987, VHDL fut adopté par IEEE comme une norme de langage de description pour le matériel.

Le langage VHDL est conçu de manière à nous permettre de modéliser des systèmes complexes décrits à des niveaux d'abstractions très différents. De plus, VHDL est un langage modulaire et hiérarchique. Un système complexe peut être divisé en plusieurs blocs, chaque bloc peut à son tour être divisé en plusieurs sous blocs et ainsi de suite. L'interface entre les différents blocs se fait par des "liens de communication". De la même manière, un modèle VHDL peut être composé de plusieurs sous-modules, chaque sous-module peut être divisé à son tour en d'autres sous-modules et ainsi de suite.

VHDL a d'abord été proposé comme un langage de modélisation et il a graduellement été utilisé comme un langage de synthèse. Cependant, il faut noter que les possibilités de modélisation avec VHDL dépassent de loin les besoins de la synthèse. Certaines caractéristiques abstraites de VHDL n'ont pas d'équivalent dans la logique digitale. Autrement dit, la synthèse ne supporte qu'un sous ensemble des possibilités de VHDL.

4.9.2.1. Flot de conception basé sur le VHDL

La Figure 4.9 [95] illustre les différentes étapes du flot de conception utilisant le langage VHDL.

Le flot de conception basé sur VHDL part d'une description du système à réaliser. Les fonctions complexes sont décrites de manière comportementale. Par exemple, un contrôleur (ou séquenceur) est décrit comme une machine d'états finis (*finite state machine*), une partie opérative comme une unité arithmétique et logique (ALU) est décrite comme un flot de données régi par des équations booléennes et contrôlé par un signal d'horloge.

Le modèle peut être écrit au moyen d'un éditeur de texte standard, mais il existe aussi des outils graphiques capables de générer du VHDL à partir d'une description sous forme de blocs (organigrammes, machines d'états, etc.).

Le modèle VHDL peut être validé par simulation. Un environnement de test (*testbench*) peut être écrit en VHDL [93]. Il déclare une instance du composant à tester et un ensemble de stimulis, ou vecteurs de test. Les fonctions du système peuvent être ainsi vérifiées avant de disposer d'une réalisation détaillée de celles-ci.

Le modèle VHDL peut être ensuite synthétisé. Un outil de synthèse est capable de transformer une description comportementale en un circuit optimisé à base de portes logiques.

L'optimisation est gouvernée par un ensemble de contraintes fournies par l'utilisateur sur la surface et/ou les délais que doit satisfaire le circuit. La synthèse se base aussi sur une bibliothèque de synthèse contenant les descriptions de toutes les cellules (portes) disponibles dans la technologie utilisée. Là aussi le format de la bibliothèque dépend de l'outil de synthèse. Le résultat de la synthèse logique est un circuit.

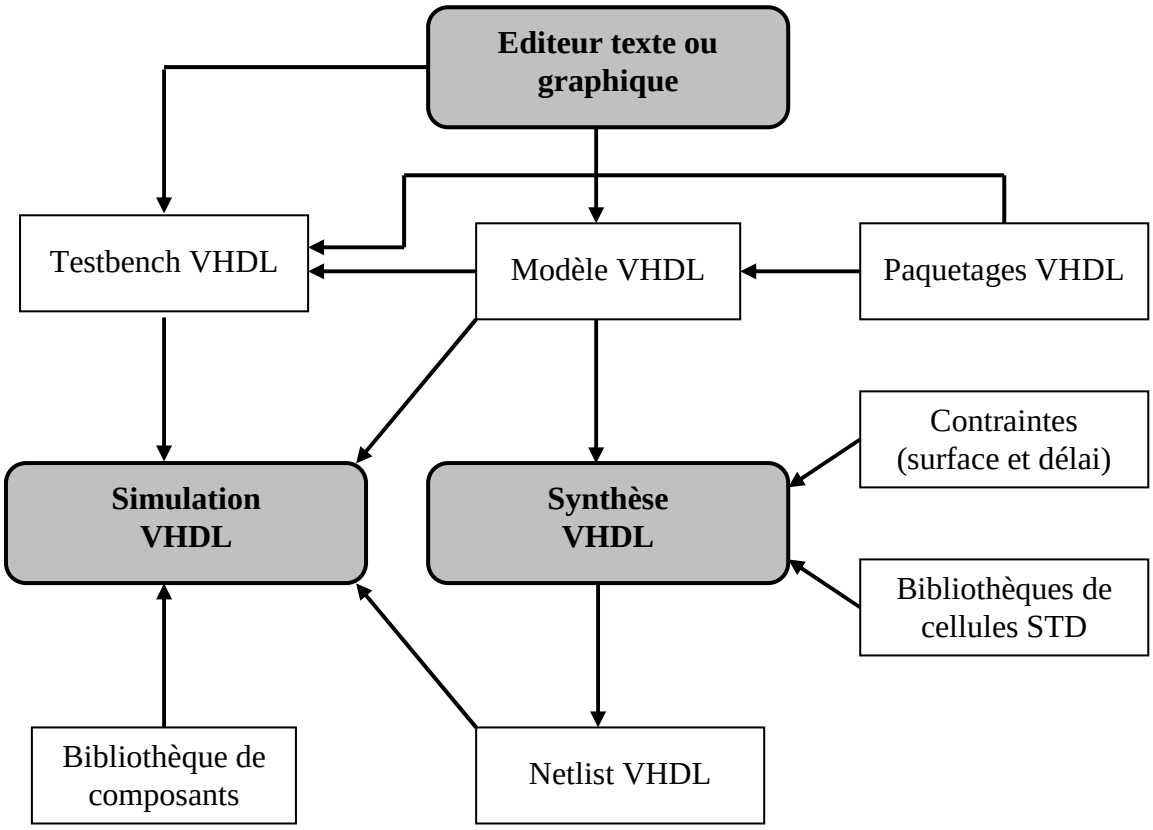


Figure 4.9. Flot de conception basé sur VHDL [95].

4.9.2.2. Organisation d'un modèle VHDL

4.9.2.2.1. Unités de conception

L'unité de conception (*design unit*) est le plus petit module compilable séparément. VHDL offre cinq types d'unités de conception:

- la déclaration d'entité (*entity declaration*);
- le corps d'architecture (*architecture body*), ou plus simplement architecture;
- la déclaration de configuration (*configuration declaration*);
- la déclaration de paquetage (*package declaration*);
- le corps de paquetage (*package body*).

Les trois premières unités de conception permettent la description de l'aspect matériel d'un système, alors que les deux dernières permettent de grouper des informations pouvant être réutilisées pour la description de plusieurs systèmes différents.

4.9.2.2.2. Entité de conception

L'entité de conception (*design entity*) est l'abstraction de base en VHDL. Elle représente une portion d'un système matériel possédant une interface entrée-sortie et une

fonction bien définie. Une entité de conception est constituée d'une déclaration d'entité et d'un corps d'architecture correspondant.

Une entité de conception peut représenter un système matériel à plusieurs niveaux de complexité: un système entier, un sous-système, une carte, un circuit intégré, une cellule complexe (ALU, mémoire, etc.), une porte logique.

Une configuration (*configuration*) permet de définir comment plusieurs entités de conception sont assemblées pour constituer le système complet. Une entité de conception peut être décrite comme une hiérarchie de composants (*component*) interconnectés. Chaque composant peut être lié à une entité de conception de plus bas niveau qui définit la structure ou le comportement de ce composant. Une telle décomposition d'une entité de conception et les liens de ses composants éventuels à d'autres unités de conception, forment une hiérarchie représentant le système complet.

4.9.3. Modélisation et simulation des modules de notre système

Etant donné que la modélisation de notre système se ferait en plusieurs modules définis individuellement, tous ces modules sont analysés et les produits de compilation sont rangés dans une bibliothèque de travail pour les réutiliser dans la modélisation des autres modules. En effet, le VHDL dissocie l'aspect externe d'un module de sa description interne. La vue externe en VHDL se traduit par une spécification d'entité, le terme entité désigne le module lui-même.

4.9.3.1. Le sélecteur

A. Le module « comparateur »

Description de la vue externe du « comparateur »

La description de la vue externe du module « comparateur » est représentée dans la figure 4.10.

Les ports se définissent de la façon suivante :

- Hor : L'horloge.
- PS_0 et L_0: sont le port de sortie et sa longueur à comparer avec les autres longueurs
- L_1, L_2,… L_K : sont les longueurs des autres ports de sortie.
- PS : est le résultat après comparaison, le résultat peut être le PS_0 entré ou bien 0.

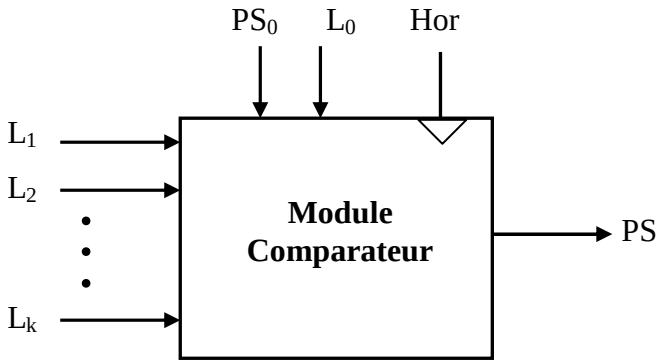


Figure.4.10. Vue externe d'un comparateur

Description de l'architecture du « comparateur »

Le but de la description interne d'un module est de décrire la façon de fonctionnement de ce module. Notre objectif dans notre étude est d'exploiter au maximum le parallélisme pour décrire les modules de notre système, et pour cela nous exploitons les instructions concurrentes offertes par le VHDL, on cherche à détecter et exploiter les points où on peut utiliser le parallélisme et / ou la simultanéité d'événements.

Dans la description du module « comparateur », on a implémenté l'opérateur de comparaison de la longueur L_0 du PS_0 avec toutes les autres longueurs des autres PSs en parallèle utilisant K processus. Chaque processus compare L_0 avec l'un des autres longueurs et donne un résultat binaire 0/1. Si les résultats de tous les processus sont des '1', la sortie de ce module vaut PS_0, sinon la sortie elle vaut 0.

Simulation du « comparateur »

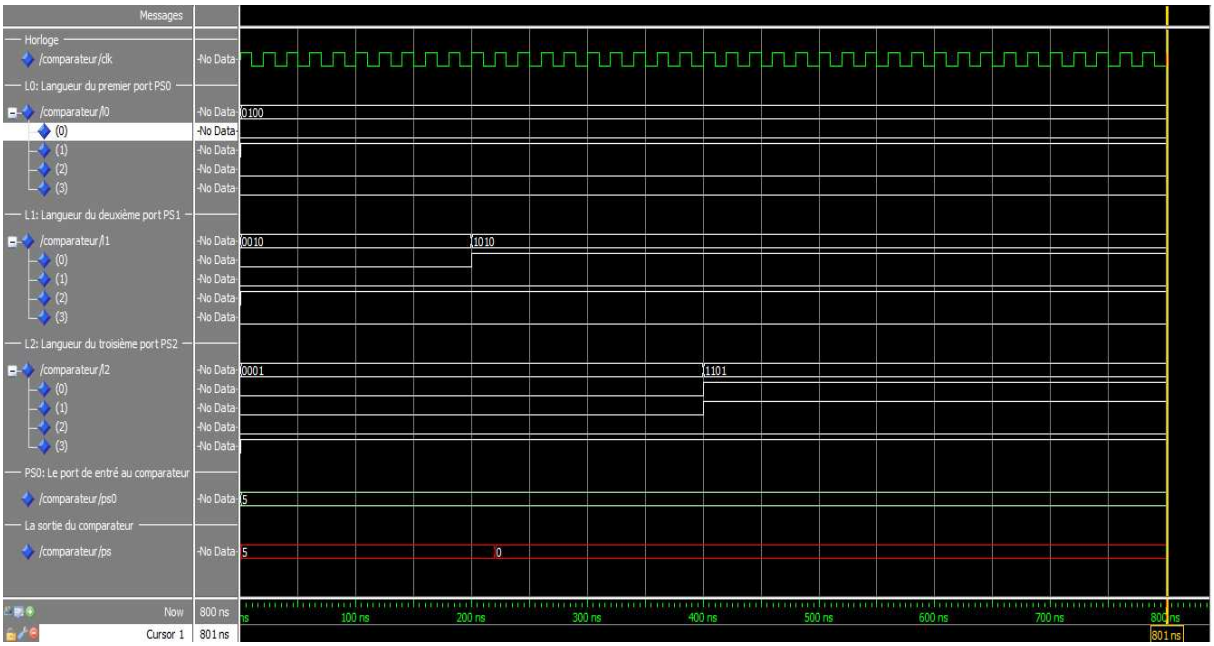


Figure. 4.11. Résultat de simulation du module « comparateur »

Interprétation des résultats

Nous pouvons voir sur cette figure que ce composant donne un résultat non nul, seulement dans la période de simulation entre 100 et 200 ns, puisque dans cette période L_0 est la plus grande longueur (L_0= '1000') et le résultat est (PS= 3) correspondant au PS_0 de longueur L_0.

B. Le module « sélecteur »

Description de la vue externe du « sélecteur »

La vue externe du module sélecteur est représentée dans la figure 4.12.

Les ports sont définis comme suit :

- Hor : L'horloge.
- PS_1, PS_2,… , PS_k : sont les ports de sorties entrés au module sélecteur.
- L_1, L_2,…, L_k : sont les longueurs des ports de sorties.
- PS : est la sortie de ce module (détermine le PS_i correspondant à la plus grande longueur.

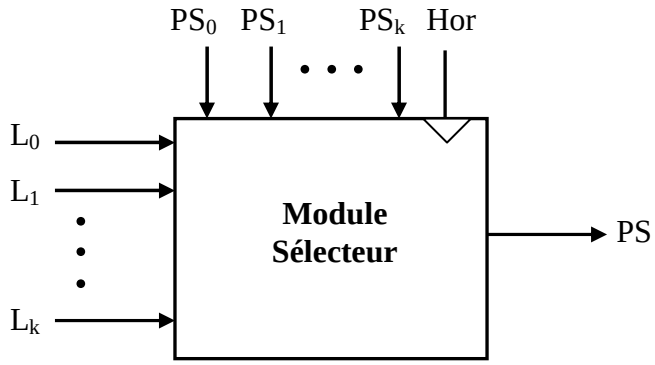


Figure.4.12. Vue externe d'un sélecteur

Description de l'architecture du « sélecteur »

Cette section montre une approche hiérarchique (modulaire) de la modélisation du module « sélecteur ». Dans un premier temps, nous décrivons une modélisation du module « comparateur » qui compare une longueur correspondant à un port de sortie (PS) avec plusieurs autres longueurs qui correspondent aux autres ports de sortie. Le comparateur donne comme résultat le port de sortie (PS) si sa longueur est supérieur à toutes les autres longueurs en entrée, ou bien un résultat nul dans le cas contraire. Ensuite, en se basant sur le module comparateur, un sélecteur travaille cette fois sur un vecteur de ports de sorties avec ses longueurs. Pour modéliser un sélecteur on utilise plusieurs instances du module comparateur pour assurer la comparaison de plusieurs ports de sorties entre eux.

Simulation du « sélecteur »

Le modèle est simulé sous le logiciel ModelSim (Figure 4.13) avec quatre unités de recherche fonctionnant en parallèle.

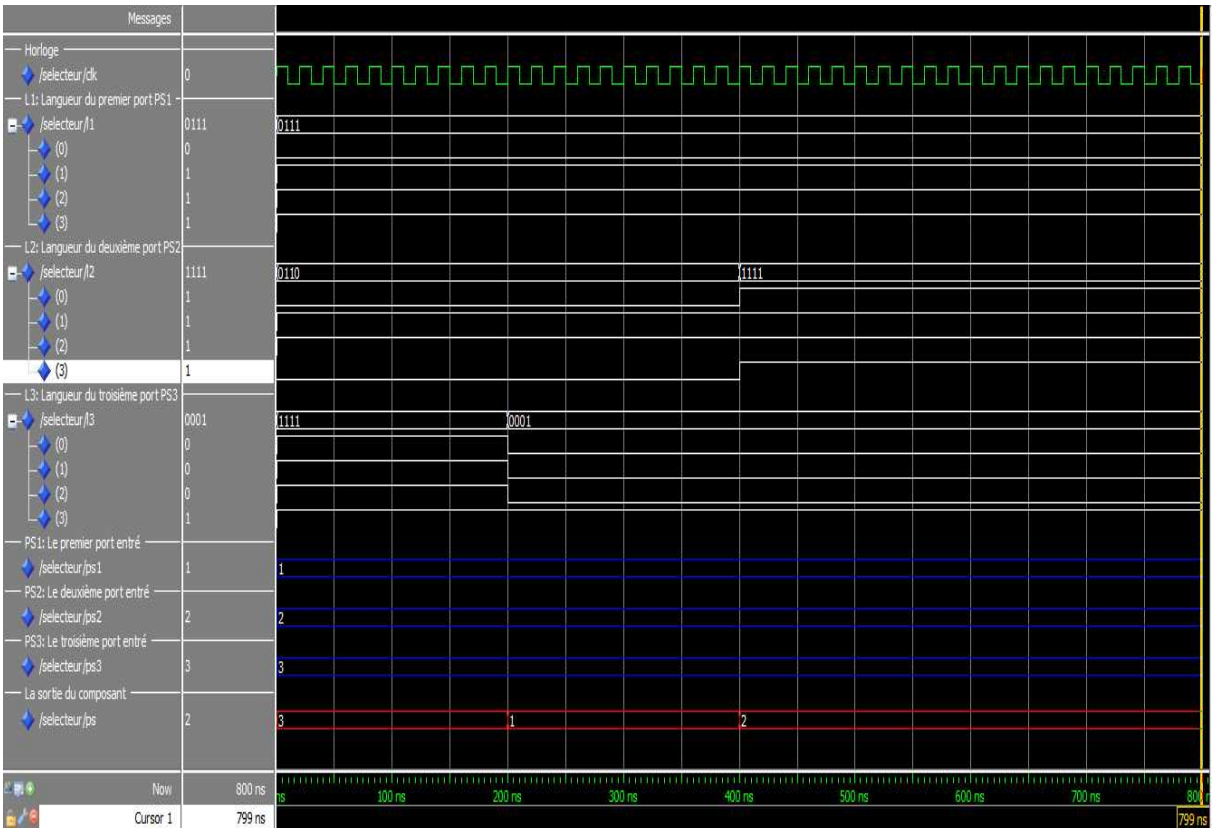


Figure. 4.13. Résultat de simulation du module « sélecteur »

Interprétation des résultats

Dans la figure de simulation, nous avons donné les valeurs suivantes aux entrées; les longueurs : L_1= '0111', L_2= '0110', L_3= '1111', et les ports de sortie : PS_1= 1, PS_2=2, PS_3=3 et nous avons mesuré la sortie PS. Les valeurs des longueurs sont en binaire, et les valeurs des ports de sorties sont en décimal.

Durant la période 0 à 200 ns, le résultat est PS= 3 qui correspond à la plus grande longueur L_3.

Durant la période 200 à 400 ns, le signal L_3 change sa valeur (L_3= '0001'), alors le résultat aussi est changé (PS = 1) et qui correspond cette fois à la longueur L_1.

Pour la période 400 à 600 ns, le signal L_2 change sa valeur (L_2= '1111'), ce qui change le résultat aussi (PS= 2).

4.9.3.2. Le composant de recherche

A. Le module « Unité de Recherche »

Description de la vue externe de « Unité de Recherche »

La description de la vue externe du module « Unité de Recherche » est représentée dans la figure 4.14.

Les ports sont définis de la façon suivante :

- Hor : L'horloge.
- Adresse_IP : l'adresse IP à chercher dans la table de routage locale de cette unité de recherche.
- PS : est le résultat après la recherche dans la table de routage, le résultat peut être le plus long préfixe correspondant à l'adresse IP entrée ou bien 0.
- L: la longueur du plus long préfixe PS.

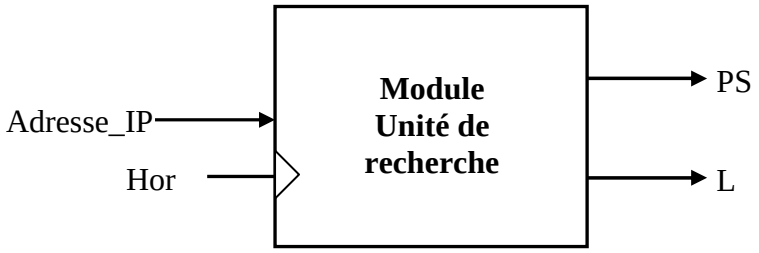


Figure.4.14. Vue externe d'une Unité de Recherche

Description de l'architecture de l' « Unité de Recherche »

Pour décrire ce module nous définissons un processus qui, d'abord sélectionne une partie de l'adresse IP (préfixe) de longueur correspondant à la longueur des préfixes de la table de routage de ce module, ensuite il cherche ce préfixe dans la table de routage, s'il existe, son port de sortie et sa longueur sont transférés au port de sortie PS du module, sinon les sorties restent des valeurs nulles.

La table de routage et la longueur des préfixes sont implémentés par des mémoires locales au module.

Simulation de « Unité de Recherche »

Pour cette simulation nous considérons la table de routage suivante :

Préfixe	Port de sortie
101011	3
111110	2
110000	4
000000	2

Tableau 4.5. Exemple d'une sous-table de routage

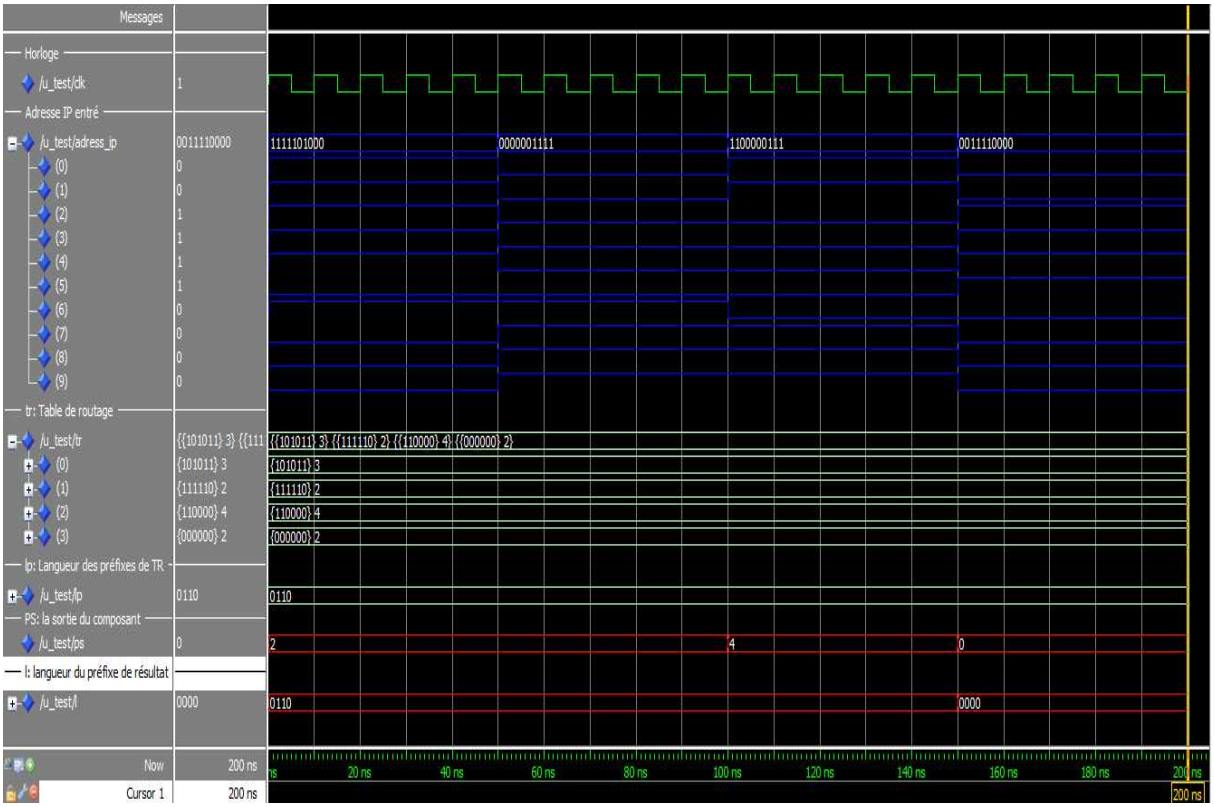


Figure. 4.15. Résultat de simulation du module « Unité de Recherche »

Interprétation des résultats

Durant la période entre 0 et 50ns l'adresse IP entré est égale à « 1111101000 », la simulation donne comme résultat (PS= 2) et (L=0110) correspondant à la deuxième entrée de la table de routage.

Durant la période entre 50 et 100ns, l'adresse IP devient « 0000001111 », ce qui donne comme résultat (PS=2) et (L=0110), correspondant à la dernière entrée de la table de routage.

Pour la période de 100 à 150ns, l'adresse IP change à la valeur « 1100000111 », le résultat est (PS= 4) et (L=0110), correspondant à la troisième entrée de la table de routage.

La période entre 150 et 200ns, l'adresse IP est égale « 0011110000 », la simulation nous donne comme résultats (PS=0) et (L=0). Pour ce dernier cas, l'unité de recherche ne trouve aucun préfixe de la table de routage qui correspond à l'adresse IP, alors notre module donne un résultat nul.

B. Le module « composant de recherche »

Description de la vue externe du « composant de recherche »

La vue externe du module composant de recherche est représentée dans la figure 4.16. Les ports sont définis comme suit :

- Hor : L'horloge.
- Adresse_IP : l'adresse IP entrée à ce module.
- PS_0, PS_1,…, PS_k : sont les ports de sortie trouvés en parallèle dans les tables de routage des unités de recherches à l'intérieur de ce module.
- L_0, L_1,…, L_k : sont les longueurs des ports de sortie.

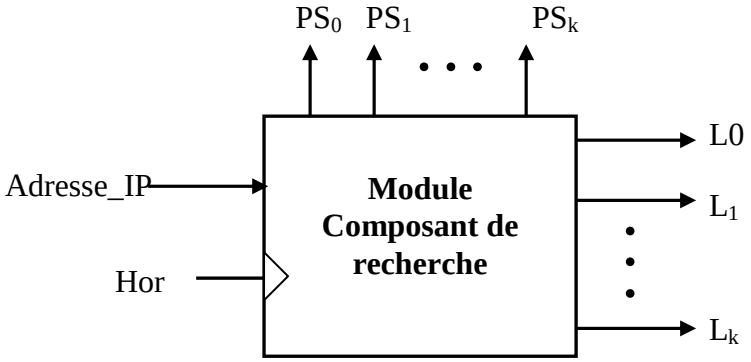


Figure.4.16. Vue externe d'un composant de recherche

Description de l'architecture du « composant de recherche »

Le module composant de recherche permet de retrouver les ports de sortie et les longueurs de leurs préfixes correspondants dans la table de routage (résultats partiels) à partir de l'adresse IP. Ce module est composé de K instances de la composante « unité de recherche » fonctionnant en parallèle. L'unité de recherche est décrite ci-dessous.

Simulation du « composant de recherche »

Paramètres de simulation

- Nombre d'unités de recherche = 3
- Adresse IP de 10 bits

Préfixe	Port de sortie
00	1
11	2
01	1
10	2

Sous-table de routage de l'unité de recherche 1

Préfixe	Port de sortie
001100	3
000111	2
000001	3
001111	4

Sous-table de routage de l'unité de recherche 2

Préfixe	Port de sortie
1100111010	3
0111000011	1
1100000000	5
0111111111	2

Sous-table de routage de l'unité de recherche 3

Tableaux 4.6. Exemple des sous-tables de routage

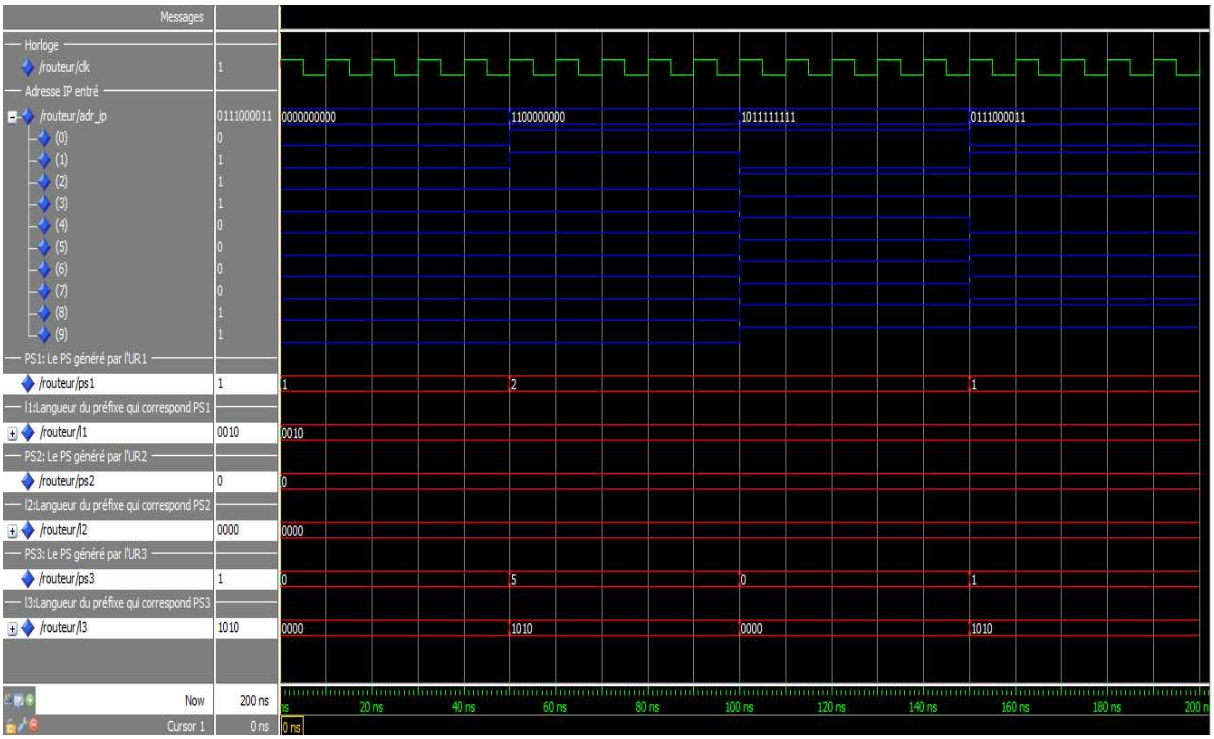


Figure. 4.17. Résultat de simulation du module « composant de recherche »

Interprétation des résultats

On distingue quatre périodes durant cette simulation, durant la période entre 0 et 50 ns, l'adresse IP est (0000000000), la simulation donne le résultat (PS_1= 1, L_1=2). Le port de sortie 1 est déterminé à partir de la table de routage qui contient des préfixes de longueur 2, (PS_2=0, L_2=0) et (PS_3=0, L_3=0) c.-à-d., les deux autres tables de routage ne contiennent aucun préfixe correspondant cette adresse IP.

Durant la période entre 50 et 100ns, l'adresse IP est (1100000000). Les résultats de simulation sont (PS_1=2, L_1=2), le port de sortie 2 est trouvé dans la table de routage qui contient des préfixes de longueur 2, (PS_2=0, L_2=0), dans la table de routage qui contient les préfixes de longueur 10, n'existe aucun préfixe qui correspond l'adresse IP. Le port de sortie 5 est trouvé dans la table de routage qui contient des préfixes de longueur 10, (PS_3=5, L_3=10).

Durant la période entre 100 et 150ns, l'adresse IP est (1011111111), la simulation donne le résultat (PS_1= 2, L_1=2) le port de sortie 2 est déterminé à partir de la table de routage qui contient des préfixes de longueur 2, (PS_2=0, L_2=0) et (PS_3=0, L_3=0). Les deux autres tables de routage ne contiennent aucun préfixe correspondant à cette adresse IP.

Durant la période entre 150 et 200ns, l'adresse IP est (0111000011), les résultats de simulation sont (PS_1=1, L_1=2), le port de sortie 1 est trouvé dans la table de routage qui contient des préfixes de longueur 2, (PS_3=1, L_3=10), le port de sortie 1 est trouvé dans la table de routage qui contient des préfixes de longueur 10, (PS_2=0, L_2=0), c.-à-d., dans la table de routage qui contient les préfixes de longueur 6, il n'existe aucun préfixe qui correspond à l'adresse IP.

4.9.3.3. Le système complet

Description de la vue externe du système complet

La vue externe du système est représentée dans la figure 4.18.

Les ports sont définis comme suit :

- Hor : L'horloge.
- Adresse_IP : l'adresse IP entrée à ce module
- PS : Le port de sortie choisi à partir des sous tables de routage des différentes unités de recherche.

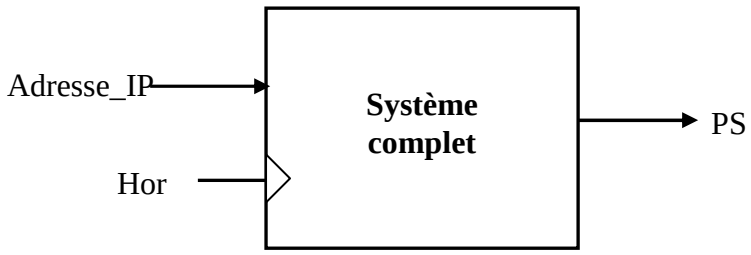


Figure.4.18. Vue externe du système complet

Description de l'architecture du « composant de recherche »

Le système complet est constitué des deux modules « composant de recherche » et « sélecteur » connectés en séquence, le sélecteur reçoit les résultats du composant de recherche et sélectionne un port de sortie.

Simulation du « composant de recherche »

Paramètres de simulation

On teste notre système en utilisant les mêmes sous-tables de routage du tableau 4.6 et les adresses IP de la figure 4.17.

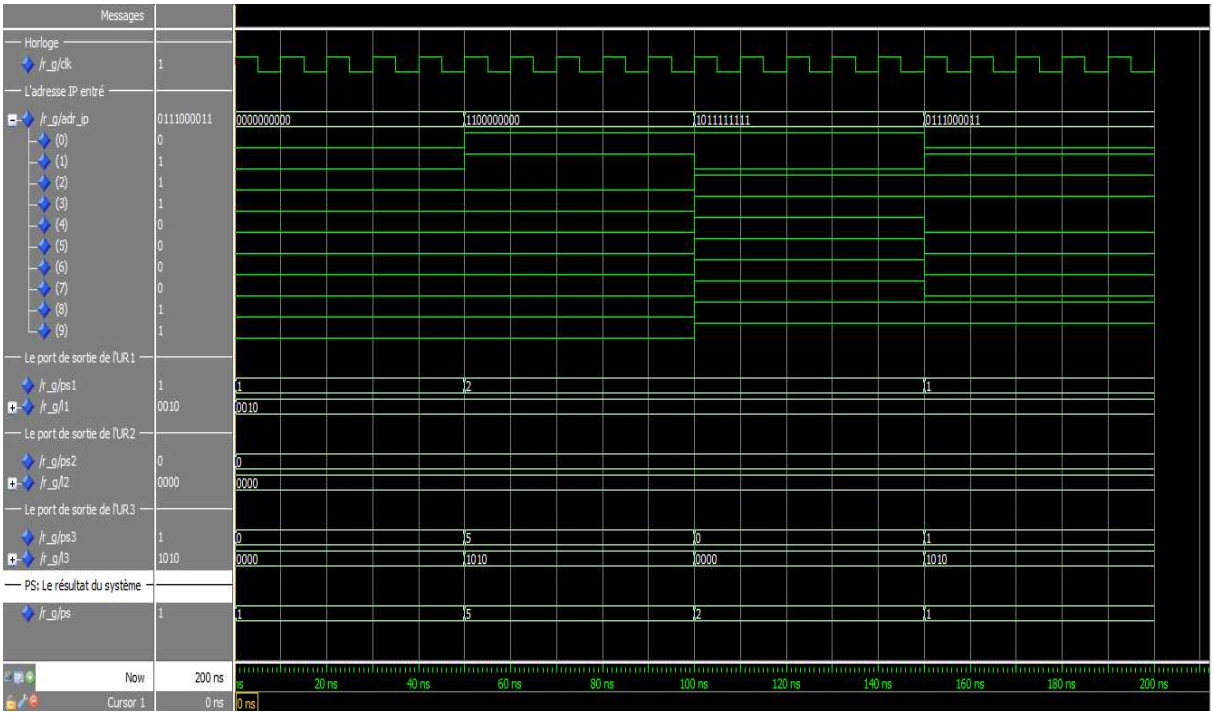


Figure. 4.19. Résultat de simulation du système complet

Interprétation des résultats

Durant la période entre 0 et 50ns, on remarque que seulement PS_1 est non nul et les autres sont nuls, alors, le résultat est ($PS=PS_1=1$).

Durant la période de 50 à 100ns, on à ($PS_1= 2$, $L_1= 2$), ($PS_2= 0$, $L_2= O$) et ($PS_3= 5$, $L_3= 10$), dans ce cas le système choisit parmi ces résultats partiels le PS correspondant au plus long préfixe (L maximal). Le résultat est ($PS= PS_3= 5$).

Pour la période entre 100 et 150, les résultats partiels sont ($PS_1= 2$, $L_1= 2$), ($PS_2= 0$, $L_2= 0$) et ($PS_3= 0$, $L_3= 0$), le système choisit le PS non nul qui est ($PS= PS_2= 1$).

Durant la dernier période, on à ($PS_1= 1$, $L_1= 2$), ($PS_2= 0$, $L_2= O$) et ($PS_3= 1$, $L_3= 10$), le système choisi parmi les PSs non nul, l'un qui correspond au plus long préfixe qui est ($PS= PS_3=1$).

4.10. Conclusion

La contribution de ce chapitre consiste en une proposition d'une solution matérielle pour améliorer la performance de l'opération de recherche d'information de routage (port de sortie d'un routeur) dans une table de routage. La solution proposée vise comme critère d'amélioration, le temps écoulé pour la recherche du plus long préfixe dans la table de routage. Cette solution est un algorithme parallèle implémenté dans une architecture parallèle.

Nous avons évalué les performances de notre solution proposée, d'une part, l'algorithme de décomposition de la table de routage en sous tables équilibrées, et d'un autre coté, la performance de la recherche en parallèle dans les différentes sous tables.

Ce volet de notre étude nous permet de conclure que les résultats obtenus démontrent la justesse des modèles retenus, sachant que cette méthode de conception permet de déceler les problèmes de vérification, puis de réduire le facteur d'erreur humaine. La description et la simulation du circuit numérique menée par l'appui de l'outil informatique VHDL. Cet outil nous a permis d'obtenir un certain niveau de réutilisabilité des différents blocs de l'architecture.

Chapitre 5

Solutions logicielles proposées

Dans ce chapitre, nous présentons deux solutions logicielles. Dans la première, nous proposons l'utilisation d'une nouvelle table cache conjointement avec la table de routage principale, et cela pour stocker les informations de routage des paquets de données récemment routés, ces informations sont utilisées pour de futures recherches. La deuxième solution est un algorithme de recherche dans l'arbre binaire des préfixes ; l'idée de notre proposition consiste à exploiter les nœuds vides de l'arbre binaire classique pour le stockage des copies des préfixes récemment utilisés par le routeur. Enfin, nous analysons les performances de ces deux solutions, et nous les comparons avec d'autres solutions trouvées dans la littérature.

5.1. Introduction

Dans la section 2 de ce chapitre, nous présentons une solution basée sur le principe de la mise en table cache des préfixes récemment manipulés afin de les utiliser pour des futures recherches. La section 3 présente notre troisième solution sous forme d'un algorithme basé sur une structure d'arbre binaire à contenu dynamique. Une étude comparative entre cet algorithme et d'autres algorithmes proposés dans la littérature prouve l'efficacité de ces solutions. Enfin, la section 4 conclut avec un résumé des contributions de ce chapitre.

5.2. Routeur avec une table de routage cache

Une table de routage cache est une table qui enregistre temporairement les adresses IP les plus récemment utilisées avec leurs informations de routage provenant de la table de routage principale, afin de diminuer le temps de recherche. La table de routage cache est une table relativement petite et rapide par rapport à la table de routage principale. La mise des adresses IP dans une table cache sert à accélérer l'opération de recherche des informations de

routage (ports de sortie), puisque la recherche se fait par des comparaisons directes entre l'adresse IP du paquet entré au routeur et les adresses IP de la table cache.

Le diagramme de la figure.5.1 schématise la structure d'une table de routage d'un routeur avec une table de routage cache. La table de routage se compose de deux tables, la table de routage principale et la table de routage cache. Quand un paquet arrive à un port d'entrée d'un routeur, le port extraira son adresse IP de destination et cherche l'information de routage dans la table de routage cache, si cette adresse IP à été trouvée, le paquet de données est expédié au port de sortie trouvé dans la table de routage cache, si l'adresse IP n'existe pas dans la table de routage cache une nouvelle recherche est déclenchée dans la table de routage principale utilisant la technique de recherche par les préfixes (le plus long préfixe) [22]. Une fois que le port de sortie correspondant au plus long préfixe de l'dresse IP est trouvé dans la table de routage principale deux opérations principales doivent exécutées:

- Une expédition du paquet de données au port de sortie,
- Une mise à jour de la table de routage cache par cette nouvelle information; insertion d'une nouvelle entrée contenant l'adresse IP, le préfixe trouvé dans la table de routage principale et le port de sortie. Ces informations peuvent être utilisées pour une future recherche dans la table de routage cache.

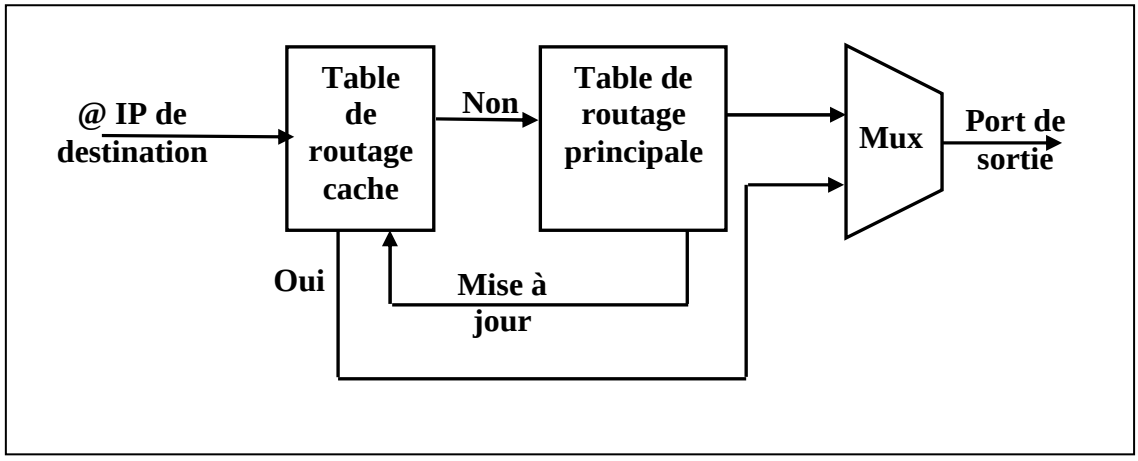


Figure.5.1. Schéma de la table de routage proposée

5.2.1. Structure de la table de routage

5.2.1.1. Table de routage principale

Plusieurs adresses destination sont agrégées en une seule entrée (préfixe) dans la table de routage principale. CIDR (*Classless Inter-Domain Routing*) [11] permet l'agrégation d'adresses contiguës.

Préfixe	Longueur du préfixe	Port de sortie
11*	2	A
10*	2	B
111*	3	C
1101*	4	D

Tableau.5.1. Exemple 5.1 d'une table de routage principale

Par conséquent en utilisant CIDR les préfixes dans la table de routage principale peuvent avoir une longueur arbitraire de 0 à 32 bits pour l'IPv4 et de 0 à 128 pour l'IPv6.

Avec cette agrégation hiérarchique la taille des tables de routage peut être réduite considérablement. Pour réduire d'avantage les tables de routage, CIDR permet des préfixes pas forcement disjoints. C'est à dire qu'à l'intérieur d'une plage d'adresses ayant la même information de routage, on peut définir une plage plus petite avec une information de routage différente. Autrement dit la plage la plus petite (préfixe plus long) représente une exception de la plage la plus grande (préfixe plus court), le tableau.5.1 est un exemple d'une table de routage principale d'un routeur. Le problème de la recherche d'information de routage dans la table de routage principale consiste donc, à trouver pour une adresse IP de destination d'un paquet de données, le plus long préfixe correspondant à cette adresse parmi l'ensemble des préfixes d'une table de routage. En raison de la complexité de l'opération de recherche du plus long préfixe dans la table de routage principale, notre proposition est d'ajouter une table de routage cache à coté de la table de routage principale pour stocker les adresses IP des paquets plus récemment routés.

5.2.1.2. Table de routage cache

Notre table de routage cache est utilisée comme la table de routage principale pour la recherche d'information de routage d'une façon rapide. Lors de la transmission des paquets IP, le routeur recherche dans la table de routage, l'entrée qui correspond à l'adresse IP du paquet de données. Pour assurer la rapidité de recherche dans la table cache on propose d'utiliser une table de taille limitée, contrairement à la table de routage principale où la recherche se fait par comparaison directe des entrées de la table cache avec l'adresse IP, donc, la structure de la table cache proposée est la suivante:

- **L'adresse IP complète** : On évite l'utilisation des préfixes dans la table cache pour accélérer la recherche par comparaison directe.
- **Le préfixe trouvé dans la table de routage principale :** Cette information est utilisée pour les opérations de mise à jour de la table de routage principale.
- **Le port de sortie (*Next-hop*)** : L'interface du routeur par laquelle sont envoyés les paquets routés.

Le tableau.5.2 représente un exemple d'une table cache avec adresses IP de 5 bits

Adresse IP	Préfixe	Port de sortie
11011	110*	A
10110	101*	B
11010	11*	C
11111	111*	D

Tableau.5.2. Exemple d'une table de routage cache

Etant donné que la taille de la table cache en nombre d'entrées est limitée pour simplifier la recherche d'information de routage, la table cache stocke les adresses IP et leurs informations de routage provenant de la table principale, le problème posé dans ce cas est que le nombre d'adresses IP est très grand par rapport à la taille de la table cache, ce qui nous oblige à utiliser le principe de remplacement des adresses IP dans la table cache.

5.2.2. Algorithme de remplacement des entrées de la table cache (la moins récemment utilisée)

Notre approche comporte non seulement l'opération de recherche d'information de routage elle même, mais aussi les opérations nécessaires pour la mise à jour de la table de routage cache. La table cache ne peut pas contenir toutes les adresses IP traitées par la table principale à cause de sa taille limitée, et pour cela il faut définir une méthode indiquant quelle adresse de la table cache doit être remplacée par la nouvelle adresse d'un paquet de donnée reçu au routeur. Cette méthode est appelée méthode de remplacement des adresses IP dans la table cache.

En ce qui concerne la restauration, une adresse IP est restaurée en table cache au moment où ses informations de routage sont recherchées et trouvées dans la table de routage principale. Si la table cache est pleine il faut retirer une adresse de la table cache. La question qui se pose à ce niveau est quelle est l'adresse qui va laisser sa place à l'adresse restaurée? L'idéale est de retirer une adresse juste après sa « dernière » utilisation (correspondant au dernier paquet d'un flux de paquets routés), mais ceci nécessite de connaître le dernier paquet des donnés pour un flux de paquets de données. Pour s'approcher au mieux de l'algorithme optimal, il faut se baser sur l'observation suivante : étant donné que tous les paquets d'un même flux de données ayant la même adresse de destination, sont transférés en général en séquence, une fois qu'un utilisateur déclenche une opération de transfert de données, les routeurs cherchent une route pour tous les paquets de données constituant son flux de données vers leur destination d'une façon continue. On peut en déduire l'algorithme suivant: quand un défaut d'adresses se produit dans la table cache, c'est l'adresse qui n'a pas été utilisée pendant le plus de temps qui est retirée et remplacée par la nouvelle adresse.

Cette méthode est appelée la moins récemment utilisée. Pour l'implanter totalement, nous avons proposé de gérer une liste chaînée de toutes les adresses IP de la table cache, avec l'adresse la plus récemment utilisée en tête et la moins récemment utilisée en queue de la liste. Cette liste doit être mise à jour à chaque référence à la table cache.

5.2.3. Ordonnancement de la table cache

A l'arrivée d'un nouveau paquet de données au routeur, ce dernier prend l'adresse IP du paquet et recherche l'information de routage dans la table cache, une fois cette adresse IP trouvée, le port de sortie correspondant dans la table cache est retenu pour router le paquet de données, et ensuite l'entrée correspondante est déplacée à la tête de la liste des adresses IP.

Par cette technique de déplacement, l'adresse IP la plus récemment utilisée est située en tête de liste et la moins récemment utilisée est située en queue de liste. A l'arrivée d'une nouvelle adresse à insérer dans la table cache, l'élément en queue de la liste sera l'élément remplacé.

Procédure d'ordonnancement :

```
Procédure Ordonnancement
{
TantQue il arrive un nouveau paquet de données
{
   Adr ← L'adresse du paquet reçu
  TantQue  la liste List_Adr_IP  est  non terminée
  {
     Si l'adresse Adr est trouvée dans la liste List_Adr_IP  Alors
      {
         Retourner le port de sortie et
         déplacer cet élément en tête de la liste  List_Adr_IP
       }
  }
}
}
```

5.2.4. Expérimentation

Pour évaluer l'efficacité de notre mécanisme proposé pour la recherche d'information de routage, nous avons développé une application qui permet de :

- Générer d'une part des exemples de tables de routage avec des tailles différentes, contenant des préfixes de longueurs différentes,
- Et d'autre part de générer des séries de paquets IP comme une trace de paquets IP, pour être routés utilisant la table de routage générée.

Nous avons utilisé pour nos expériences une base de données représentant la table de routage, l'objectif du premier test est de varier le nombre d'entrées de la table cache pour préciser la taille optimisant le temps de recherche pour une série de 496965 paquets IP, en utilisant une table de routage de 10000 préfixes. Pour l'évaluation de l'opération de recherche

des informations de routage, la table de routage n'a subi ni suppressions, ni insertions de préfixes tout au long de l'expérimentation.

Taille de la table cache	**Période écoulée pour le routage**	**Taille de la table cache**	**Période écoulée pour le routage**
5	1866	500	1306
10	1883	700	1207
50	1751	800	1161
100	1660	900	1144
150	1589	1000	1089
300	1440	10000	807

Tableau.5.3. Période de routage en fonction de la taille du cache

Le tableau.5.3 représente les résultats obtenus, cependant, nous avons remarqué que, quand on augmente le nombre d'entrées de la table de routage cache, le temps de recherche d'information de routage des paquets IP est diminué comme montre la figure.5.2.

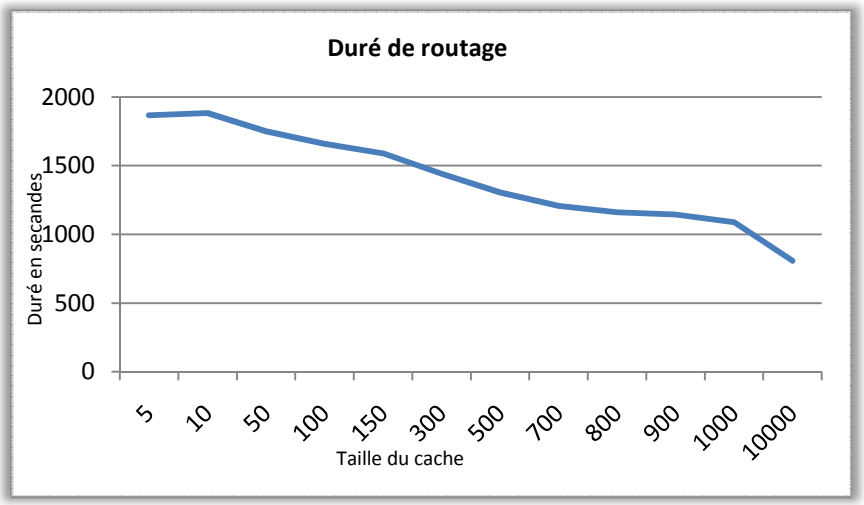


Figure.5.2. Temps de recherche en fonction de la taille du cache

Dans un deuxième test, nous comparons notre proposition avec l'algorithme de recherche d'information de routage par un arbre binaire classique [13] sur quatre séries de paquets IP.

Nous avons utilisé une table de routage de 10000 préfixes et une table de routage cache de 0.1% de la taille de la table de routage principale. Le tableau.5.4 représente les durées écoulées pour la recherche des informations de routage des quatre séries de paquets IP.

Séries de paquets IP	Routage par cache	Routage par arbre binaire [13]
2345	100	375
22349	500	5028
226588	850	1978
450700	3939	12000

Tableau.5.4. Comparaison de notre algorithme avec l'algorithme de recherche par arbre binaire classique

La figure.5.3 montre les distributions de temps de recherche pour les quatre séries de paquets IP testées. Il est clair que notre algorithme améliore la durée de recherche des informations de routage pour les quatre séries de paquets IP.

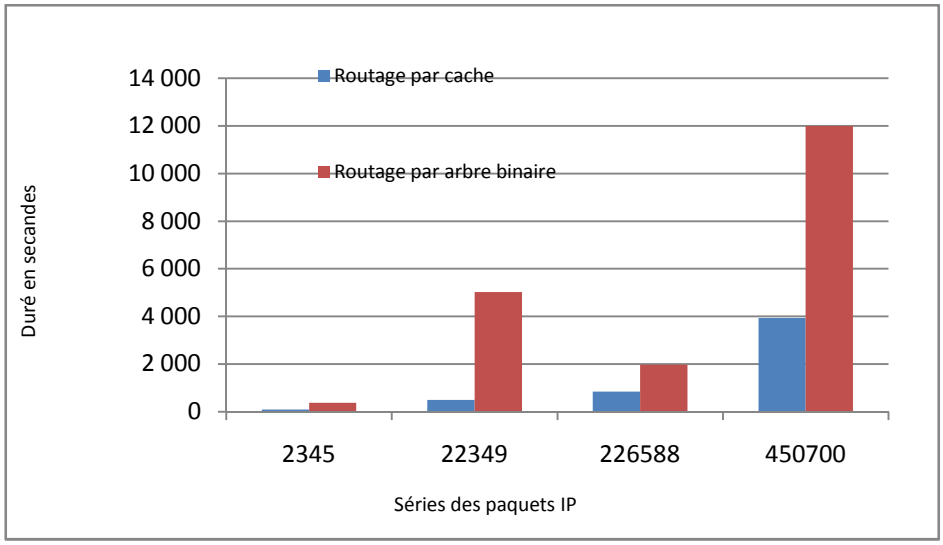


Figure.5.3. Distributions de temps de recherche d'information de routage pour les quatre séries de paquets IP

5.3. Algorithme de recherche d'information de routage par arbre binaire à contenu dynamique (ABCD)

Dans cette partie, un algorithme de recherche de plus long préfixe en utilisant un arbre binaire à contenu dynamique (ABCD) est proposé. Notre algorithme basé sur le principe d'exploitation des nœuds internes vides de l'arbre binaire classique des préfixes, les nœuds vides sont utilisés pour le stockage des préfixes les plus récemment utilisés. Les résultats d'évaluation de performances montrent que notre algorithme est efficace en termes de nombre moyen d'accès mémoire, puisque, la recherche est terminée lorsque l'adresse IP correspond à un préfixe dans un nœud interne d'origine vide sans arriver à un nœud feuille.

5.3.1. Description de l'arbre binaire à contenu dynamique (ABCD)

L'arbre binaire à contenu dynamique est la structure de données utilisée pour le stockage des préfixes de la table de routage. Le contenu de cet arbre est variable selon les adresses IP traitées par le routeur. La technique proposée sert à exploiter les nœuds vides de l'arbre binaire classique de préfixes [12] [13] et le placement des copies de préfixes récemment utilisés dans les nœuds vides des niveaux supérieurs de l'arbre, de tel sorte que l'opération de recherche du plus long préfixe se termine dès que le plus long préfixe correspondant à l'adresse IP du paquet de données est trouvé dans un certain nœud interne dans l'arbre binaire sans arriver obligatoirement à une feuille. Le tableau.5.5 est un simple exemple d'une table de routage, les chaînes binaires (préfixes) de la première colonne du tableau sont représentées par les nœuds noirs dans l'arbre de la figure.5.4, les nœuds blancs sont des nœuds vides c.-à-d.qu'ils ne contiennent pas de préfixes.

Préfixes	Longueurs	Ports de sortie
00*	2	A
010*	3	B
101*	3	C
1*	1	D
111*	3	E
111111*	6	F
110100*	6	G
110101*	6	H
111100*	6	I
1100*	4	J

Tableau.5.5. Exemple 5.2 d'une table de routage

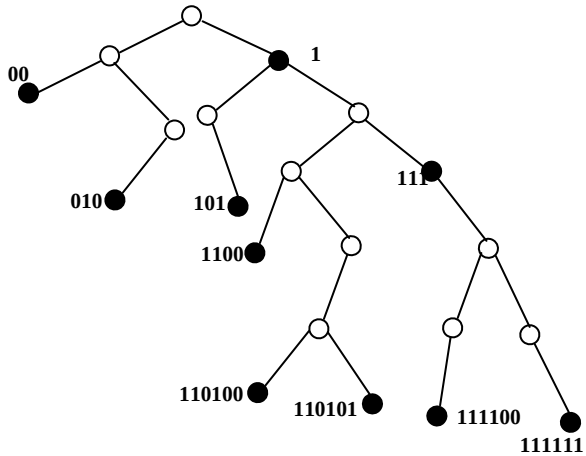


Figure.5.4. Arbre binaire de préfixes construit à partir du Tableau 5.5

L'arbre (ABCD) proposé est construit au fur et à mesure de l'opération de routage des paquets IP, son contenu est dynamique c.-à-d. qui change selon les adresses IP traitées par le routeur. Soit comme exemple les adresses IP de huit bits suivantes : 01001111, 10100111, 11111110, 00010001, 11010001, 01001101, 11010110, 00011100, après la recherche des plus long préfixes de ces adresses IP, le contenu de l'arbre binaire de la figure.5.4 est modifié comme représente l'arbre de la figure.5.5, des nœuds vides dans l'arbre de la figure.5.4 deviennent des nœuds contenant des copies de préfixes des niveaux inférieurs de l'arbre binaire.

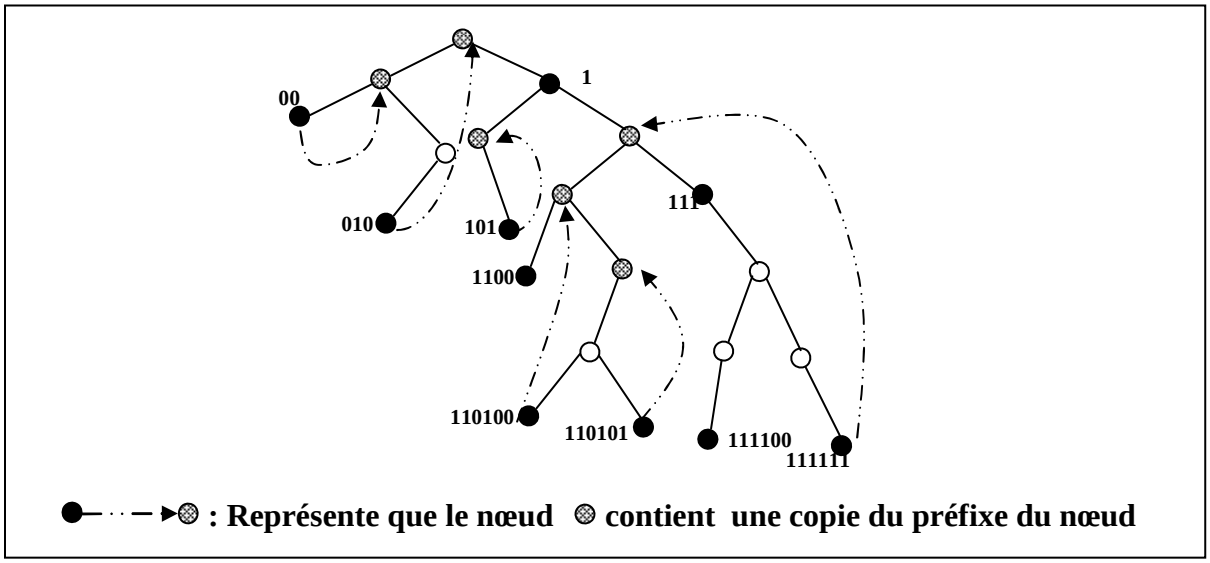


Figure.5.5. L'arbre binaire à contenu dynamique (ABCD) proposé

Selon l'arbre de la figure.5.4, le plus long préfixe correspondant à l'adresse « 01001111 » est 010*, alors une copie du préfixe 010* est stockée dans le nœud vide du plus haut niveau dans le chemin de ce préfixe à partir de la racine de l'arbre, cette opération est effectuée avec toutes les adresses IP de l'exemple.

Dans l'arbre dynamique de la figure.5.5, on distingue trois types de nœuds :

- Les nœuds noirs sont des nœuds originalement contenant des préfixes
- Les nœuds blancs sont des nœuds vides
- Les nœuds avec motif qui sont des nœuds originalement vides, mais actuellement contiennent des copies de préfixes déjà manipulés.

5.3.1.1. Structure des nœuds de l'arbre ABCD

La figure.5.6 illustre la structure d'un nœud de l'arbre proposé. Tout nœud est de type 1, 2, 3 ou 4. Un nœud de type 1 contient un préfixe d'origine de l'arbre binaire initialement construit, un nœud de type 2 ou 3 est un nœud d'origine vide, mais il contient temporellement une copie d'un préfixe stocké dans un autre nœud interne de niveau inférieur,

un nœud de type 3 est un nœud aussi d'origine vide, mais il contient temporellement un préfixe d'une feuille, un nœud de type4 est un nœud vide (sans préfixe).

Type	Préfixe	PS	Fils G	Fils D	PPO

Figure.5.6. Structure d'un nœud de l'arbre ABCD

- **Type** = 1, 2,3 ou 4
- **Préfixe** : Le préfixe de la table de routage
- **PS** : Le port de sortie correspondant au préfixe dans la table de routage.
- **Fils G, Fils D** : Pointent vers le fils gauche/droit respectivement de ce nœud, ou bien Nil.
- **PPO** : Si le nœud est de type 1, ce pointeur pointe vers le nœud qui contient une copie de ce préfixe ou bien Nil; si le nœud est de type 2 ou 3, ce pointeur pointe vers le nœud d'origine de ce préfixe, ce pointeur est Nil si le nœud est de type 4.

5.3.1.2. Construction de l'arbre ABCD

L'arbre ABCD est construit initialement comme un arbre binaire classique qui contient tous les préfixes de la table de routage [12], un préfixe d'une entrée de la table de routage définit un chemin dans l'arbre binaire début à partir de la racine et se termine par un nœud de l'arbre. Toutes les feuilles de cet arbre contiennent des préfixes, mais les nœuds internes peuvent contenir des préfixes ou peuvent rester vides s'il n'y a pas de préfixe qui se termine au niveau de ce nœud comme le montre l'exemple de la figure.5.4. Tous les nœuds sont marqués comme contenant des préfixes ou bien des nœuds vides, le contenu de cet arbre binaire est modifié lors de l'exécution de l'opération de recherche du plus long préfixe correspondant comme suit : Le principe de notre proposition est d'exploiter les nœuds vides de l'arbre binaire pour stocker des copies des préfixes, les plus récemment utilisés. Une fois un paquet IP arrive, le routeur cherche dans l'arbre binaire, le plus long préfixe correspondant à son adresse IP. Pour accélérer les futures recherches pour cette même adresse IP, notre algorithme stocke une copie de ce préfixe dans un nœud vide de haut niveau de son chemin à partir de la racine de l'arbre. Trois cas sont à considérer à ce niveau :

- le premier cas, s'il existe un nœud vide dans le chemin du préfixe, une copie de ce préfixe est stockée dans ce nœud et il est marqué, comme nœud de type2 si le nœud d'origine du préfixe est une feuille dans l'arbre binaire, sinon le nœud est marqué comme un nœud de type3.
- Le deuxième cas est l'absence de nœuds vides dans tous le chemin de ce préfixe, s'il existe un nœud d'origine vide mais contient une copie d'un autre préfixe (nœud de type2 ou 3), ce dernier est remplacé par le nouveau préfixe.

- Le troisième cas s'il n'existe pas de nœuds de type 2, de type 3 ou de type 4 dans le chemin du préfixe, aucune copie de ce préfixe n'est stockée dans un autre nœud.

5.3.2. Algorithme de recherche du plus long préfixe correspondant dans l'arbre ABCD

Quand un paquet de données arrive au routeur, l'adresse de destination (notée Adr) du paquet est extraite et l'algorithme Recherche_route(Adr, T) de la figure.5.7 est exécuté. Le paramètre *T* est un pointeur vers la racine de l'arbre ABCD. La procédure de recherche se termine immédiatement lorsque le préfixe stocké dans un nœud de type 4 correspond à une sous chaine de l'adresse IP ou bien la recherche arrive à une feuille de l'arbre.

```
Algorithme Recherche_route(Adr, T)
{
TantQue  (T ≠ Nil )
{
Case Type_noeud (T)
1: {   PS ←PS(T) ;  NCP ← T   Approfondir (T)   } ;
2: {   NOV← T ;
            Si Get_Sous_Chaine (Adr,longueur(Prefix(T)) = Prefix(T) Alors
               PS ← PS(T) ;  T← PPO(T) //Branchement vers le préfixe d'origine
            Sinon    Approfondir (T) ;
   }
3: {  NOV← T ;
            Si Get_Sous_Chaine (Adr,longueur(Prefix(T)) = Prefix(T) Alors
               PS ← PS(T) ;
               T← Nil ; // Plus long préfixe trouvé
            Sinon    Approfondir (T)
   };
4: {   NAV← T ;  Approfondir (T)   }
} //Fin de case
} // Fin Tant que
// Copier le préfixe dans un nœud de haut niveau
Si PPO (NCP) ≠ Nil Alors
{
     Si NAV ≠ Nil Alors
     {
      Copier (NAV, NCP)
     }
      Sinon
     {
            Si NOV ≠ Nil Alors
```

```
            {
              Copier (NOV, NCP)
             }
        }
        Si NCP est feuille Alors
          Type_noeud(NAV) ← 3
        Sinon
          Type_noeud(NAV) ← 4
}
Return PS
}
```

Figure.5.7. Algorithme de recherche du plus long préfixe pour une adresse IP

La fonction *Get_Sous_Chaine(Adr, L)* retourne une sous chaîne de longueur *L* à partir de l'adresse IP notée Adr ; La procédure *Copier(x,y)* copie les données du nœud y dans le nœud x de l'arbre. La fonction *Approfondir (T)* sert à approfondir dans l'arbre par un niveau selon le bit actuel de l'adresse IP, soit au fils gauche si le bit est 0 ou bien au fils droit dans le cas contraire.

5.3.3. Expérimentation et résultats

Pour évaluer les performances de notre algorithme, nous avons utilisé différentes tailles de tables de routage générées aléatoirement sous forme de bases de données. Etant donné que l'accès à la mémoire est l'opération qui prend beaucoup de temps dans le processus de recherche dans la table de routage, c.-à-d. le temps écoulé par le reste des traitements est négligé devant le temps des accès à la mémoire, pour cela nous avons évalué la performance de recherche du plus long préfixe seulement en termes de nombre d'accès à la mémoire.

	Np	**NTn**	**Nnv**	**Npd**
Table 1	47011	2494824	2445755	2058
Table 2	184940	9631910	9441841	5149
Table 3	457360	5706370	5240521	8489

Tableau.5.6. Performances de recherche dans l'arbre ABCD

Le tableau.5.6 représente l'évaluation de l'arbre ABCD après le routage d'une série de 4947145 paquets IPV6 utilisant trois tables de routage table1, table2 et table3 contenant respectivement 47011, 184940 et 457360 préfixes, l'arbre est évalué en termes de :

- Nombre de préfixe (Np)
- Nombre total de nœuds (NTn)
- Nombre de nœuds vide (Nnv)
- Nombre de préfixes déplacés à un niveau supérieur dans l'arbre (NPd).

Comme montre le tableau.5.6, plus de 1% de préfixes sont déplacés à un niveau supérieur dans l'arbre ce qui accélère l'opération de recherche du plus long préfixe dans l'arbre ABCD. Le nombre de préfixes déplacés dans l'arbre est en fonction des adresses IP traitées, et il peut l'augmenter jusqu'à arriver à un arbre sans nœuds vides.

Tables de routage		**Table de routage 1**	**Table de routage 2**	**Table de routage 3**
Nombre de préfixes		47011	184940	457360
Arbre binaire [12]	$\mathbf{T_{moy}}$	24,96	26,03	31,64
	$\mathbf{T_{max}}$	42	45	48
Arbre de priorité [27]	$\mathbf{T_{moy}}$	29,8	33,84	36,39
	$\mathbf{T_{max}}$	37	43	46
Arbre ABCD proposé	$\mathbf{T_{moy}}$	21,14	22,43	24,17
	$\mathbf{T_{max}}$	62	66	58

Tableau.5.7. Comparaison des performances avec d'autres algorithmes

Le tableau.5.7 illustre la comparaison des performances de l'arbre ABCD proposé avec l'arbre binaire classique [12] et l'arbre de priorité [27] en termes de nombre moyen d'accès mémoire (T_{moy}) et de nombre maximum d'accès mémoire (T_{max}). L'arbre binaire à contenu dynamique proposé présente une bonne performance en termes de nombre moyen d'accès mémoire, puisque le processus de recherche dans notre proposition s'arrête dès qu'il trouve une copie d'un préfixe feuille dans certain nœud interne. Par contre, le nombre maximum d'accès mémoire se trouve très grand dans notre proposition, à cause de l'opération supplémentaire de stockage des copies des préfixes dans des nœuds vides de niveaux supérieurs dans l'arbre.

5.4. Conclusion

La contribution de ce chapitre a consisté en la proposition des solutions logicielles pour améliorer la performance de l'opération de recherche d'information de routage (port de sortie correspondant au plus long préfixe) dans une table de routage. La première solution sert à ajouter une nouvelle table de routage appelé table cache, cette table stocke les adresses IP récemment manipulées dans la table de routage principale, de façon que tous les futurs paquets qui correspondent aux mêmes informations de routage trouveront leurs informations de routage dans la table cache. La deuxième solution s'est basée sur une structure de donnée arborescente (arbre ABCD); l'algorithme exploite les nœuds vides dans l'arbre binaire des préfixes pour le stockage des préfixes récemment recherchés.

Nous avons évalué et comparé les performances des mécanismes proposés à celles d'autres systèmes proposés dans la littérature. Le constat est que nos solutions améliorent la vitesse de recherche d'information de routage dans la table de routage.

Chapitre 6

Conclusion et perspectives

L'Internet est un réseau à commutation de paquets où chaque paquet est traité indépendamment de tous les autres. La performance de l'Internet dépend directement de la capacité de traitement des paquets de données par les routeurs. Un routeur est un équipement d'interconnexion de réseaux informatiques permettant d'assurer le routage des paquets entre deux réseaux, ou plus, afin de déterminer le chemin qu'un paquet de données va emprunter. Pour cela, le routeur exécute une opération de recherche d'information de routage dans sa table de routage. Plusieurs mécanismes permettant d'améliorer la performance de l'opération de recherche d'information de routage dans les routeurs Internet ont été proposés dans cette thèse. Nous la concluons avec un résumé des idées développées et des contributions dans ce cadre. Ensuite, nous discutons les orientations pour les travaux futurs.

Afin de mieux comprendre les différentes solutions pour le problème de recherche d'informations de routage (*adresses lookup*) dans une table de routage des routeurs IP, nous avons proposé dans cette thèse une classification de ces solutions, et qui peuvent être classées comme des solutions matérielles ou logicielles.

Beaucoup d'approches matérielles de manipulation des adresses IP et des tables de routage ont été proposées par les chercheurs. Ces solutions sont en général des mémoires associatives CAMs et TCAMs avec différentes architectures.

Des architectures et mécanismes de mise des préfixes dans des mémoires cache sont proposées pour le stockage des informations de routage, essentiellement celles qui ont été récemment utilisées afin de les exploiter pour des futures recherches. Notre deuxième solution est basée sur ce principe de stockage des informations de routage, mais avec une solution logicielle proposée à côté des mémoires cache matérielles.

Nous avons exploité des techniques de parallélisme, dont l'utilisation, devient une nécessité dans les architectures dédiées aux traitements de données au niveau des routeurs

Internet. Cette approche peut effectivement répondre aux besoins des routeurs actuels à cause de l'agrandissement des tailles des réseaux de télécommunication. Les architectures SIMD s'adaptent mieux aux opérations de traitement de données au niveau des routeurs à cause des quantités de données passées et traitées par les routeurs.

Une architecture parallèle dédiée à l'opération de recherche du plus long préfixe dans une table de routage a été aussi proposée dans cette thèse. Cette proposition consiste à décomposer la table de routage en plusieurs sous tables équilibrées et manipulées en parallèle. Pour la décomposition de la table de routage, on s'est basé sur l'affectation des préfixes appartenant à un intervalle de longueurs, à une même sous table ; les préfixes d'une telle sous table sont expansés ou étendus à des préfixes de même longueur afin de simplifier la recherche des informations de routage dans les différentes sous tables par une simple comparaison.

Nous nous sommes intéressés aussi à l'évolution des différentes techniques logicielles de recherche d'information de routage pour l'acheminement des paquets de données, et des différentes structures de données proposées pour le stockage des préfixes d'une table de routage. La majorité des solutions logicielles sont des algorithmes utilisant les structures arborescentes pour la représentation des préfixes de la table de routage, à savoir, des arbres représentant les bits des différents préfixes (représentation binaire). Ces solutions se basent aussi sur la transformation et la représentation des préfixes sous forme de plages d'adresses IP (arbres d'adresses IP), et sur la représentation des valeurs des préfixes dans des arbres binaires de recherche. Nous avons montré que ces structures ne sont pas toujours optimales et présentent souvent des inconvénients, ce qui a motivé la proposition de notre troisième solution.

Enfin nous pouvons situer les contributions principales de cette thèse dans des propositions visant à améliorer la vitesse de l'opération de recherche d'informations de routage dans les routeurs IP, et ce, dans le but d'accélérer d'avantage l'opération d'acheminement des paquets de données sur Internet.

Travaux futurs

La performance du processus d'acheminement de paquets de données dans les routeurs est essentielle pour la performance de l'Internet. Même si nous avons proposé des mécanismes, pour améliorer les performances de recherche d'information de routage dans les routeurs IP, de plus en plus les routeurs demandent d'autres améliorations de leurs performances. La poursuite de ce travail offre des perspectives dans plusieurs axes.

Nous nous sommes focalisés dans la première proposition sur l'amélioration du critère de vitesse de recherche d'information de routage, beaucoup plus que sur l'espace mémoire occupé par les sous tables de routage de notre architecture, surtout après la transformation des préfixes par l'opération d'expansion dans les sous tables de routage. Un objectif qui reste à accomplir consiste en l'amélioration de cette proposition en termes d'espace mémoire occupé par les sous tables de routage.

Concernant la simulation de notre architecture matérielle proposée, nous avons modélisé et décrit l'architecture en utilisant le langage de description de matériel VHDL, avant de tester le modèle obtenu en utilisant l'outil Modelsim. Pour la réalisation matérielle, il reste à passer à la phase de synthèse du circuit en générant l'architecture au niveau bas (niveau du composant électronique).

Des points importants comme les mises à jour des tables de routage n'ont pas été abordés dans les trois solutions que nous avons proposées; nous pensons qu'il reste primordial de pouvoir les étudier et les évaluer dans les conditions réelles. Une prochaine étape de cette étude peut prendre en compte ces opérations de mise à jour.

Bibliographie

[1] http://www.apps.ietf.org/rfc/rfc791.html

[2] F. Baker, " Requirements for IP Version 4 Routers ", *Internet RFC 1812*, 1995.

[3] http://www.ietf.org/rfc/rfc790.txt

[4] http://www.ietf.org/

[5] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *RFC 2460*, 1998.

[6] R. Hinden et S. Deering, " Internet Protocol Version 6 (IPv6) Addressing Architecture," *RFC 3513*, 2003.

[7] http://www.faqs.org/rfcs/rfc2373.html

[8] http://www.cidr-report.org/

[9] http://www.faqs.org/rfcs/rfc1338.html

[10] Y. Rekhter et T. Li, "An architecture for IP address allocation with CIDR." *RFC 1518*, http://www.ietf.org/rfc/rfc1518.txt, September 1993.

[11] V. Fuller, T. Li, J. Yu et K. Varadhan. " Classless inter-domain routing (CIDR): an address assignment and aggregation strategy", *RFC 1519, http:/ /www.ietf.org/rfc/rfc1519.txt*, September 1993.

[12] E. Fredkin, "Trie Memory ", *ACM*, Vol. 3, Issue. 9, pages 490–500, 1960.

[13] D. Antos, " Overview of Data Structures in IP Lookups " *CESNET Tchnical Report* 9/2002.

[14] P. Gupta, S. Lin et N. McKeown, « Routing lookups in hardware at memory access speeds », *INFOCOM (3) IEEE*, Vol.3, pages 1240-1247, 1998.

[15] M. Degermark, A. Brodnik, S. Carlsson, et S. Pink, « Small forwarding tables for fast routing lookups ». *ACM SIGCOMM Computer Communication Review*, Vol.27, Issue. 4, pages 3-14, 1997, Jul 1997.

[16] A. Moestedt et P. Sjodin. « IP address lookup in hardware for high speed routing ». *Hot Interconnects VI*, August 1998.

[17] N. F. Huang et S. M. Zhao. « A novel ip-routing lookup scheme and hardware architecture for multigigabit switching routers ». *IEEE JSAC: IEEE Journal on Selected Areas in Communications*, Vol. 17, Issue. 6, pages 1093 - 1104 , June 1999.

[18] S. Nilsson et G. Karlsson, « IP-Address Lookup Using LC-Tries », *IEEE Journal pn selected areas in communications"*, Vol.17, Issue. 6, Pages 1083 – 1092, 1999.

[19] V. Srinivasan et G. Varghese. « Fast address lookups using controlled prefix Expansion ». *Proceedings of ACM Sigmetrics*, Vol. 17, Issue. 1, pages 1-40, June 1999.

[20] M. Á, R. Sánchez et W. Dabbous, « Un mécanisme optimisé de recherche de route IP », *INRIA*, 2004.

[21] V. Srinivasan et G. Varghese, "Fast IP lookups using controlled prefix expansion,"

ACM Transactions on Computer Systems, Vol. 26 Issue. 1, pages 1-40, June 1998

[22] M. R. Sanchez, E. Biersack, et W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE Network Magazine*, Vol. 15 Issue. 2, March/April 2001

[23] S. Nilsson and G. Karlsson, "Fast Address Look-Up for Internet Routers", *Proeding of.*

IEEE Broadband Commununication. '98, pages. 9-18, 1998.

[24] Ravikumar V.C, R. Mahapatra et J.C. Liu, « Modified LC-Trie Based Efficient Routing Lookup ", *Proceedings of the 10th IEEE Int.l Symp. on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS.02)*, pages 177 – 182, 2002.

[25] D.R. Morrison. "PATRICIA — practical algorithm to retrieve information coded in alphanumeric", *ACM*, Vol. 15, N°. 14, pages 514-34, October 1968.

[26] K. Sklower, "A Tree-Based Packet Routing Table for Berkely UNIX," *Technical report,*

University of California, Berke. 1999.

[27] H. Lim et J. Mun, "An efficient IP address lookup algorithm using a priority-trie ", *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE,* pages 1-5, 2006.

[28] X. Sun et Y. Q. Zhao, "An On-Chip IP Address Lookup Algorithm", IEEE Trnasactions on computers, VOL. 54, NO. 7, pages 873 - 885 , JULY 2005.

[29] H. Lu et S. Sahni, « O(log n) Dynamic Router-Tables for Prefixes and Ranges» *IEEE* Trnasactions on computers, VOL. 53, N°. 10, pages 1217-1230, OCTOBER 2004

[30] S. Suri, G. Varghese, et P. Warkhede, " Multiway Range Trees: Scalable IP Lookup with Fast Updates", *IEEE, Global Telecommunications Conference, 2001. GLOBECOM '01*. vol. 3 , pages 1610 - 1614 , 2001.

[31] Y.K. Li et D. Pao, "Address lookup algorithms for IPv6", *IEE Proceding on Communication*, Vol. 153, Issue. 6, pages909 – 918, December 2006

[32] B. Lampson, V. Srinivasan, et G.Varghese, " IP Lookups Using Multiway and Multicolumn Search ", Proc. *IEEE/ACM Networking, Transactions*, Vol. 7, Issue. 3, pages 1248-1256, 1999.

[33] H. Lu et S. Sahni, "A B-Tree Dynamic Router-Table Design", *IEEE Computers Transactions* , Vol.54, Issue:7, pages.813-823, 2005

[34] N. Yazdani et Paul S. Min, " Fast and Scalable schemes for the IP address Lookup Problem", Proceedings *of the IEEE Conference on High Performance Switching and Routing*, Vol.3 pages 1610 - 1614, 2000.

[35] C. Yim, B. Lee, et H. Lim, " Efficient binary search for IP address lookup ", *IEEE Communications Letters*, Vol. 9, N°. 7, pages 652-654, 2005

[36] H. Lim, B. Lee, et W. J. Kim, " Binary searches on multiple small trees for IP address lookup", *IEEE Communications Letters*, Vol. 9, N°. 1, pages 75-77, 2005

[37] H. Lim, W. Kim, et B. Lee, " Binary search in a balanced tree for IP address lookup ", *Proceding of IEEE HPSR2005*, pages 490-494, 2005,

[38] L.C Wuu, T.J Liu, et K.M Chen, "A longest prefix first search tree for IP lookup". *Computer Networks,* Vol. 51, Issue. 12, pages 3354–3367, August 2007.

[39] H. Lim et H. G. Kim, " IP Address Lookup for Internet Routers using Balanced Binary Search with Prefix Vector ", *IEEE* Transactions on communications, Vol. 57, N°. 3, pages 618-621, 2009

[40] M. Kobayashi, T. Murase, et A. Kuriyama, "A Longest Prefix Match Search Engine for

Multigigabit IP Processing ", *IEEE International Conference on Communications,* vol.3 , pages 1360-1364, 2000

[41] V.C. Ravikumar, et R. N. Mahapatra, " TCAM architecture for IP lookup using prefix properties ", *IEEE*, Vol. 24, Issue. 2, pages 60-69, 2004

[42] R. Panigrahy et S. Sharma, " Reducing TCAM Power Consumption and Increasing

Throughput " , *2002. IEEE 10th Symposium* of *High Performance Interconnects*, pages 107-112, 2002.

[43] Z. Wang, H. Che,M. Kumar, et S.K. Das, CoPTUA, " Consistent Policy Table Update Algorithm for TCAM without Locking ", *IEEE Transactions on Computers*, Vol. 53,N°. 12, pages 1602-1614, 2004.

[44] F. Zane, G. Narlikar et A. Basu, " CoolCAMs: Power-Efficient TCAMs for Forwarding Engines", *INFOCOM . Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE*, Vol.1, pages 42 – 52, 2003.

[45] T. Mishra et S.Sahni, " PETCAM – A Power Efficient TCAM For Forwarding Tables", IEEE Symposium on Computers and Communications, pages 224 - 229, 2009.

[46] D. Shah et P. Gupta, " Fast Updating Algorithms on TCAMs", *IEEE Micro,* Vol 21, Issue 1, pages 36-47, 2001.

[47] H. Liu, " Routing Table Compaction in Ternary-CAM", *IEEE Micro*, Vol. 22, Issue 3, Pages 58 – 64, 2002.

[48] V.C. Ravikumar, R. N. Mahapatra, et L. N. Bhuyan, " EaseCAM: An Energy And Storage Efficient TCAM-Based Router Architecture for IP Lookup", *IEEE Transactions on Computers*, Vol. 54, N°. 5, pages 521-533, 2005.

[49] M. J. Akhbarizadeh, et M. Nourani, " Hardware-Based IP Routing Using Partitioned Lookup Table", IEEE/ACM transactions on networking, Vol. 13, N°. 4, 2005.

[50] N. Mohan, et M. Sachdev, " Low Power Dual Matchline Ternary Content Addressable Memory", *IEEE International Symposium on Circuits and Systems*, 2004.

[51] H. Miyatake, M. Tanaka, et Y.Mori, "A design for high-speed low-power CMOS fully parallel content addressable memory macros", *IEEE Journal of Solid State Circuits*, Vol. 36, N°. 6, pages 956-968, 2001.

[52] C.S. Lin, J.-C. Chang, et B.-D Liu, "A low-power pre-computation based fully parallel content addressable memory", *IEEE Journal of Solid State Circuits*, Vol. 38, N°. 4, pages 654-662, 2003.

[53] W. Lu et S. Sahni, " Low Power TCAMs For Very Large Forwarding Tables", *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 316 – 320, 2008.

[54] G. Wang et N. Tzeng, " TCAM-Based Forwarding Engine with Minimum Independent Prefix Set (MIPS) for Fast Updating ", *IEEE International Conference of Communications*, pages 103-109, 2006.

[55] T. Mishra et S. Sahni, " DUO–Dual TCAM Architecture for Routing Tables with Incremental Update ", *IEEE Symposium on Computers and Communications (ISCC)* , pages 503 – 508, 2010.

[56] B. Talbot, T. Sherwood, et B. Lin, " IP caching for terabit speed routers". *IEEE Conference on Global Telecommunications, GLOBECOM '99*, Vol. 2, pages 1565–1569, 1999.

[57] I.L Chvets et M.H MacGregor, " Multi-zone caches for accelerating IP routing table lookups", *IEEE Workshop on High Performance Switching and Routing*, pages 121–126, 2002.

[58] S. Kasnavi, P. Berube, V. Gaudet, et J.N Amaral, "A cache-based Internet Protocol address lookup architecture", *IEEE, Computer Networks*, Vol. 52, Issue 2, pages 303–326, 2008

[59] T.C Chiueh et P. Pradhan, " High-performance IP routing table lookup using CPU caching", *IEEE INFOCOM*, Vol 3, pages 1421–1428, 1999.

[60] W.L Shyu, C.S Wu, etT.C Hou, " Multilevel aligned IP prefix caching based on singleton Information", *IEEE Conference on Global Telecommunications, GLOBECOM '02.* , Vol. 3, pages 2345–2349, 2002.

[61] M.H MacGregor, " Design algorithms for multi-zone IP address caches" *IEEE Workshop on High Performance Switching and Routing*, pages 281–285, June 2003.

[62] H. Liu, " Reducing cache miss ratio for routing prefix cache" *IEEE Conference on Global Telecommunications, GLOBECOM '02* , Vol. 3, pages 2323–2327, 2002.

[63] R.L Rudell, " Multiple-valued logic minimization for PLA synthesis". *Technical Report*

UCB/ERL M86/65, EECS Department, University of California, Berkeley, 1986.

[64] K. Shiomoto, M. Uga, M. Omatani, S. Shimizu, et T. Chamaru, " Scalable Multi-Qos IP

ATM switch router architecture" *IEEE Communications Magazine*, Vol. 38, pages 86–92, 1999.

[65] L. Peng, W. Lu, et L. Duan, « Power efficient IP lookup with supernode caching", *IEEE GLOBECOM*, Vol. 48, pages 215–219, 2007.

[66] W. Eatherton, G. Varghese, et Z. Dittia, " Tree bitmap: hardware/software IP lookups with incremental updates", ACM SIGCOMM Computer Communication Review, Vol. 34, pages 97–122, 2004.

[67] M. Akhbarizadeh et M. Nourani, " Efficient prefix caching for network processors" *IEEE Symposium on High Performance Interconnects*, pages 41–46, 2004.

[68] B. Chen et R. Morris, " Flexible Control of Parallelism in a Multiprocessor PC Router ", *Proceedings of the USENIX 2001 Annual Technical Conference*,, 2001.

[69] C. Partridge, P.P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A.King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W. C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober et G. D. Troxel, " A 50Gbps IP Router ", *IEEE/ACM Transactions. Networking*, Vol. 6, N°. 3, pages 237-248, 1998.

[70] Y. Luo, L. N. Bhuyan, " Shared Memory Multiprocessor Architectures for Software IP Routers", IEEE transactions on parallel and distributed systems, Vol. 14, N°. 12, pages 1240-1249, 2003.

[71] M. Waldvogel, G. Varghese, J. Turner, et B. Plattner, " Scalable High Speed IP Routing Lookups", *ACM Special Interest Group on Data Communications (SIGCOMM)*, pages 25-36, 1997

[72] K. J. Schultz, " *CAM-Based Circuits for ATM Switching Networks* », Thèse PhD, Université de Toronto, 1996.

[73] M. J. Flynn, « Some computer organizations and their effectiveness" *IEEE trans. On Computers*, Vol. 21, pages 948-960.

[74] J. B. Dennis, "Data flow supercomputer", *IEEE Trans, on Computers*, Vol. C13, N°. 11, pages. 48-56. 1980.

[75] R. Duncan, "A surver of parallel coputer architectures", *IEEE Trans, on Coputers*, 1990.

[76] Thinking Machine Corporation. *CM-2, Technical summary*, 1987.

[77] K. E. Batcher, "Design of a massively parallel processor" *IEEE trans. On Computers*, Vol. 29, N°. 9, 1980.

[78] "The connection CM-200 series technical summary", *Cambridge, Massachusetts: Thinking Machine Corporation*, 1991.

[79] J. F. Palmer, "The NCube family of high performance parallel computer systems", *Hypercube Concurent Computers and Applications*, Vol. 1, pp. 847-851, 1988.

[80] A. Trew et G. Wilson, "Past, Present, Parallel A Survey of Available Parallel Computing Systems », *Sringer-Verlag* , 1991

[81] P. H. Enslow, "Multiprocessor organization", *ACM Computer Surveys*, Vol. 9, 1977.

[82] H.T. Kung, "Why systolic architectures", *Computers*, Vol. 15, N°. 1, pp. 1982, pp. 37-46.

[83] H. T. Kung, "On the implementation and use of systolic array processors", *IEEE International Conference on Computer Design*, pp. 370-373, 1984.

[84] P. Qinton et Y. Robert, « Algorithmes et architectures systoliques », *Edition masson, Collection études et recherches en informatique*, 1989.

[85] A. Herscovici, « Introduction aux grands ordinateurs scientifiques », *Editions Eyrolles*, 1986.

[86] D. Etiemble, « Architecture des processeurs RISC », *Edition Armand Colin, Collection Acquis Avancées de l'informatique*, 1991.

[87] J. B. Dennis, « Mdeling the weather with a data flow supercomputer », *IEEE Trans, on Computers*, Vol. C39, N°, 7, pp. 592-603, 1991.

[88] S. Y. Kung, "Wavefront array processors: langage, architecture and applications", *IEEE Trans, on computers*, Vol. C31, N° 11, pp. 1054-1065, 1982.

[89] J. L. Roch et D. Trystram, « Méthodologie pour la Programmation Efficace d'Applications Parallèles », *La Lettre du Transputer*, Vol.6, N°.4, pp. 133-138, 1994.

[90] P. Naish et P. Bishop "Conception des Asics", Masson 1990.

[91] M. Keating et P. Bricaud "Reuse Methodology Manual For System On Chip Designs", Kluwer Academic Publishers 1999.

[92] IEEE," IEEE Standards Collection - Software Engineering", Institute of Electrical and Electronics Engineers, New York, 1994.

[93] J. Bergeron, "Writing Testbenches: Functional Verification of HDL Models", Kluwer Academic Publishers, 2000.

[94] A.A. Jerraya, H. Ding, P. Kission, et M. Rahmouni « Behavioral Synthesis and Component Reuse with VHDL », Kluer Academic Publishers, 1997.

[95] P. J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufman Publishers, 2nd edition, 2001.

[96] J. Weber et M. Meaudre « VHDL du langage au circuit, du circuit au langage » Masson

[97] IEEE Press, " Formal Verification of Commercials ICs", IEEE Design & Test of Computers, Vol 18, N°. 4, New York, 2001.

Résumé

Le réseau Internet est composé de nœuds et de routeurs reliés ensemble par des médias de transmission (liens). Les paquets de données transitent sur les liens, d'un routeur à un autre, vers leurs destinations finales. Chaque routeur exécute une décision d'expédition sur les paquets entrants pour déterminer le prochain routeur du paquet.

Le routeur Internet est un dispositif permettant de choisir le chemin que les paquets de données vont emprunter pour arriver à leurs destinations. Pour ce faire, le routeur effectue une recherche dans la table de routage en utilisant l'adresse de destination d'un paquet IP comme une clé ; cette tâche du routeur est appelée décision de routage. La croissance du nombre d'utilisateurs d'Internet a comme conséquence l'augmentation des tailles des tables de routage des routeurs et la complexité de l'opération d'expédition des paquets de données.

La décision de routage consiste à chercher la meilleure route pour un paquet à partir de son adresse IP destination. Afin d'augmenter l'efficacité du routage dans l'Internet, plusieurs adresses destination sont agrégées en une seule entrée dans une table de routage (préfixe). Depuis l'avènement du CIDR (*Classless Inter-Domain Routing*), les préfixes dans la table de routage ont des longueurs variables, alors plusieurs préfixes peuvent être associés à une même adresse IP de destination, le routeur doit choisir le plus long préfixe correspondant (PLP) à l'adresse IP. Etant donné que les préfixes dans la table de routage ont des longueurs différentes, alors, nous ne pouvons pas trouver la correspondance en utilisant un algorithme de recherche avec une correspondance exacte. Par conséquent, les performances du routeur dépendent fortement de l'efficacité de l'opération de recherche du plus long préfixe dans la table de routage.

Différentes approches logicielles et matérielles ont été proposées et déployées pour améliorer les performances des routeurs, évaluées principalement en termes de débit ou de nombre de paquets expédiés par seconde. Dans cette thèse, nous avons proposé trois mécanismes de recherche du plus long préfixe correspondant à une adresse IP dans une table de routage, les mécanismes que nous avons proposés ont contribué à améliorer la performance de l'acheminement de paquets dans les routeurs IP.

Mots clés:
Routeur IP, Table de routage, plus long préfixe, architecture parallèle, description VHDL, algorithme parallèle, expansion des préfixes, table de routage cache, arbre binaire, arbre binaire à contenu dynamique.

ملخـــص

تتكون شبكة الانترنت من مجموعة عقد وأجهزة توجيه مترابطة فيما بينها بواسطة روابط,حزم البيانات تمر عبر الروابط من موجه إلى أخر وصولا إلى وجهاتهم النهائية فكل جهاز توجيه ينفذ قرار إرسال الحزم الواردة إليه وذلك لتحديد جهاز التوجيه المقبل لكل حزمة.

جهاز التوجيه في الانترنت هو جهاز يستخدم لاختيار المسار لحزم البيانات لإيصالها إلى وجهاتهم النهائية و للقيام بذلك يقوم جهاز التوجيه بعملية بحث في جدول التوجيه باستعمال عنوان وجهة حزمة البيانات IP كمفتاح وتسمي هذه المهمة بقرار التوجيه. إن تزايد عدد مستخدمي الانترنت ينتج عنه زيادة حجم جداول التوجيه في أجهزة التوجيه وتعقيد عملية شحن حزم البيانات.

قرار التوجيه هو السعي للحصول علي أفضل طريق لحزمة المعلومات باستعمال عنوان الوجهة ولزيادة كفاءة التوجيه في الانترنت يتم تجميع عدة عناوين في مدخل واحد في جدول التوجيه (بادئة) . منذ ظهور البروتوكول CIDR أصبحت البادئات متغيرة الأطوال إذن عدة بادئات قد توافق نفس عنوان الوجهة IP ولهذا يجب على جهاز التوجيه أن يختار أطول بادئة مطابقة للعنوان IP. باعتبار أن للبادئات في جدول التوجيه أطوال مختلفة فلا يمكن العثور على معلومات التوجيه باستخدام خوارزمية بحث بالتطابق التام, ولذلك فعالية جهاز التوجيه تعتمد اعتمادا كبيرا على فعالية عملية البحث على أطول بادئة في جدول التوجيه.

قد تم اقتراح عدة تقنيات برمجية والية لتحسين أداء أجهزة التوجيه ,حيث أن مقياس الأداء الأساسي هو التدفق أو عدد الحزم الموجهة في الثانية. في هذه الأطروحة اقترحنا ثلاثة آليات بحث عن أطول بادئة موافقة لعنوانIP في جدول التوجيه, هذه الآليات المقترحة تساهم في تحسين عملية توجيه حزم المعطيات لدى أجهزة التوجيه.

كلمات البحث:
جهاز التوجيه IP، جدول التوجيه، أطول بادئة، الهندسة المتوازية ، وصف VHDL، خوارزمية متوازية، توسيع البادئة، جدول التوجيه المخبأ ، شجرة ثنائية، شجرة ثنائية ذات المحتوى الديناميكي.